LOAD BALANCING IN SWITCH NETWORKS

A Dissertation
Presented to
The Academic Faculty

By

Sen Yang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2018

LOAD BALANCING IN SWITCH NETWORKS

Approved by:

Dr. Jun Xu, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. Henry L. Owen
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Ellen W. Zegura
School of Computer Science
Georgia Institute of Technology

Dr. Gee-Kung Chang
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Siva Theja Maguluri
School of Industrial and Systems Engineering
Georgia Institute of Technology

Date Approved: December 13, 2017

To Guirong, Benchang and Xiaolan.

ACKNOWLEDGEMENTS

I would like to express my utmost gratitude to my advisor Jun (Jim) Xu, for providing me an opportunity to work in his group and continuously supporting my study and research. His knowledge, personality, and passion inspired and helped me overcome many difficulties. What I have learnt from him is a precious asset of my life. I would like to say special thanks to my former ECE advisors, Dr. John A Copeland and Dr. George F Riley, for their kind guidance and consistent support during my Ph.D. study.

I would like to express my sincere gratitude to my other thesis committee members, Dr. Henry L. Owen, Dr. Ellen W. Zegura, Dr. Gee-Kung Chang and Dr. Siva Theja Maguluri for being great teachers and mentors, and for their interests in my graduate work and their valuable and insightful comments.

I would like to express my special gratitude to Dr. Justin Romberg, Dr. Bill Lin (UCSD) and Dr. JG (Jim) Dai (Cornell Univ) for their kind insightful guidance and hard work on our collaborated works.

I would like to thank my previous mentors in my previous internships, He Yan, Zihui Ge and Dongmei Wang (AT&T Labs); Rui Wang, Junlan Zhou, Sam Burnett and Benjamin Jones (Google), for providing opportunities to learn and broaden my horizons and support throughout these years.

My thanks also goes to current and former lab members, Liang Liu, Long Gong, Weijun Ding, Yimeng Zhao, Zhenglin Yu, Samantha Lo and all others. Thank you for the help and discussions. I am also indebted to my friends, Qianao Ju, Cen Lin, Shiyang Chen, Kai Ying, Mu Xu, Qiang Hu, Yue Zhang, Ming Yi, Qi Zhou, and many others. Thank you for creating a beautiful and unforgettable memory for me in the past years.

I would like to thank all my student co-authors, especially Shenglin Zhang (Tsinghua Univ) and Pin Yin (UCSD), for their tremendous helps and for our friendships.

I owe my deepest gratitude to my family, especially my parents, Benchang Yang and

Guirong Sun, and my girlfriend Xiaolan Wang. Their endless love, encouragement, and support are always the source of my courage and motivation. This dissertation is dedicated to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Load-balanced switch (LBS) is a promising class of switch architecture which has received wide attention due to its inherent scalability properties in both size and speed. These scalability properties continue to be of significant interest due to the relentless exponential growth in Internet traffic. The main drawback of the load-balanced switch is that packets can depart out-of-order from the switch, which can significantly degrade network performance by negatively interacting with TCP congestion control. Hence, a large body of subsequent work has proposed a variety of modifications for ensuring packet ordering, but almost all the proposed approaches tend to increase packet delay significantly in comparison to the basic load-balanced switch.

The object of this dissertation research is to explore and analyze advanced high-performance load-balanced switch architectures that can scale well in both switch size (in terms of the number of switch ports) and link speed, provide throughput guarantees, achieve low latency, and maintain packet ordering.

We investigated several methodologies to achieve this object. The first approach we considered is to rectify the packet reordering problem by simply buffering and re-sequencing the out-of-order packets at the switch outputs. Our solution is based on the following observation of the queuing dynamics in the basic LBS: although two "back-to-back" packet arrivals belonging to the same *switch flow* (i.e., they share the same input and output) may reach the output port out of order, the amount of reordering is quite limited. In particular, we formally bound the worst-case amount of time that a packet has to wait in these output reordering buffers before it is guaranteed to be ready for in-order departure with high probability, and we prove that this bound is *linear* with respect to the switch size. This linear bound is significant because previous approaches can add quadratic or cubic delays to the load-balanced switch. In addition, we use a hash-grouping method that further reduces resequencing delays significantly.

The second approach we considered is based on randomization. To prevent harmful effects in TCP performance due to out-of-order packets, only packets belonging to the same application flow (e.g. a TCP/IP flow) have to depart from their output port in order. Randomized load-balancing of application flows by means of hashing on the packet header is a well-known simple solution to this packet reordering problem in which all packets belonging to the same application flow are routed through the same intermediate port and hence the same path through the switch. Unfortunately, this method of load-balancing can lead to instability, depending on the mix of flow sizes and durations in the group of flows that gets randomly assigned to route through the same intermediate port. In our study, we show that the randomized load-balancing of application flows can be enhanced to provably guarantee both stability and packet ordering by extending the approach with *safety* mechanisms that can *uniformly diffuse* packets across the switch whenever there is a build-up of packets waiting to route through the some intermediate port.

Although simple and intuitive, our experimental results show that our output packet re-sequencing approach and our extended randomized load-balancing approach outperforms existing load-balanced switch architectures.

CHAPTER 1

INTRODUCTION

1.1 Motivation and Background

Network traffic across the Internet as well as inside data centers continues to grow exponentially. This relentless traffic growth is fueled by an increasing adoption of video streaming and cloud computing, and a proliferation of network-connected devices with increasing networking capabilities. To keep up with the ever increasing traffic demands with reliable service, network operators need high-performance switch architectures that can scale well in both switch size (in terms of the number of switch ports) and link speed, provide throughput guarantees, achieve low latency, and maintain packet ordering. Unfortunately, conventional switch architectures have not been able to keep up with these challenges.

A promising class of highly scalable switch architectures, first introduced by Chang et al. [1, 2], and later further developed by others (e.g. [3, 4, 5, 6, 7, 8]), is the *load-balanced switch* (LBS). As shown in Fig. 1.1, a generic LBS relies on two switching stages for forwarding packets. The first switching stage connects the input ports to the center stage of intermediate ports, and the second switching stage connects the center stage of intermedi-

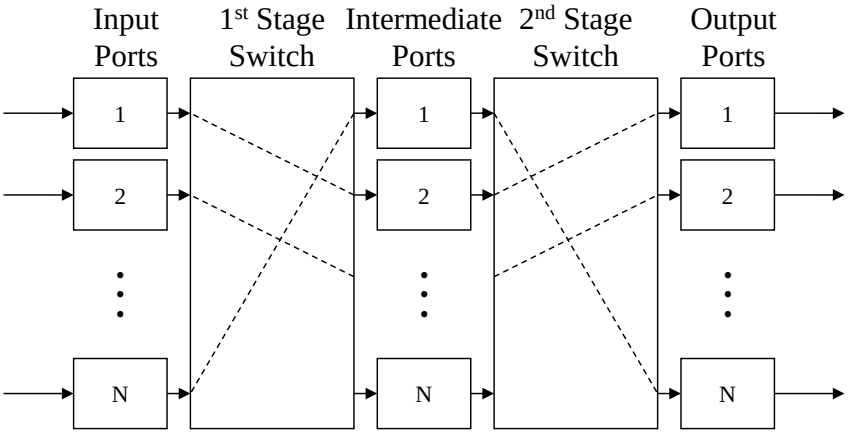


Figure 1.1: Generic load-balanced switch.

ate ports to the final stage of output ports. Both switching stages execute a deterministic connection pattern such that each input is connected to each output of a switching stage $1/N$-th of the time.

This architecture can be implemented using two identical $N \times N$ crossbar switching stages where each switching stage goes through a predetermined periodic sequence of cyclic-shift connection patterns such that each input is connected to each output of a switching stage exactly once every N cycles. Alternatively, as shown in [5], the deterministic connection pattern can also be efficiently implemented using optics where all inputs are connected to all outputs of a switching stage in parallel at a rate $1/N$-th of the line rate. LBSes are much more scalable, in terms of both switch size and link speed, than conventional switch architectures. This is because, in LBS architectures, the connection pattern during every switching cycle at each switching stage is fixed and requires zero computation, and each port can forward packets in a fully distributed manner based only on local information.

Although the basic LBS originally proposed in [1] is highly scalable, it has the critical problem that packets can depart out-of-order from the switch. In the basic LBS, consecutive packets at an input port are spread to all N intermediate ports upon arrival. These packets, going through different intermediate ports, may encounter different queuing delays. Thus, some of these packets may arrive at their output ports out-of-order. This is detrimental to Internet traffic since the widely used TCP transport protocol falsely regards out-of-order packets as indications of congestion and packet loss. The outcome is the retransmission of packets, often multiple times, further exacerbating the problem. Therefore, a number of researchers have subsequently explored this packet ordering problem.

Most existing approaches to the packet ordering problem are based on some form of complete or partial aggregation of packets into frames or stripes. Uniform Frame Spreading (UFS) [5], Full-Order Frames First (FOFF) [5], Padded Frames (PF) [4], and Sprinklers [3] are representative examples of such approaches. However, these methods pay a significant

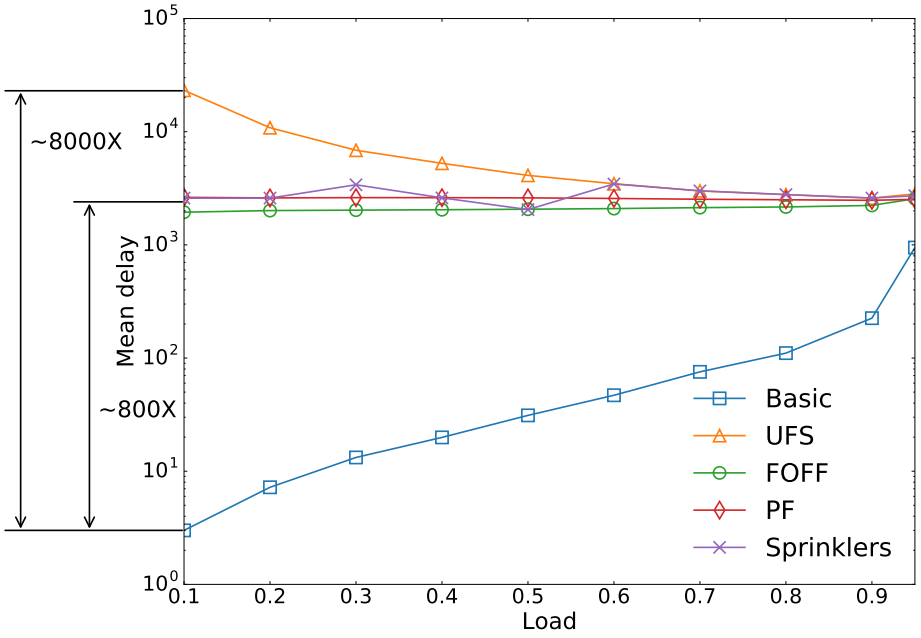


Figure 1.2: Poor average delays under moderate loads. A uniform traffic pattern was used here. The switch size $N = 64$. Loads are normalized to 1. At a load of 0.1, the average delay of the proposed schemes are about 800 times higher than the basic LBS (labeled "Basic").

price for ensuring packet ordering in that they perform significantly worse than the basic LBS [1].

Fig. 1.2 compares these aggregation-based methods with the basic LBS (labeled as "Basic") that does not guarantee packet ordering. The results are for a uniform traffic pattern, one in which the output port destination for every arriving packet is chosen uniformly at random. As can be seen, these approaches all have packet delays can be significantly higher than the basic LBS, especially under moderate loads.

1.2 Research Objectives and Main Contributions

The object of this dissertation research is to explore and analyze advanced high-performance load-balanced switch architectures that can maintain packet ordering while still achieve low latency and provide throughput guarantees. We investigated several methodologies to achieve this object.

The first approach we considered is to rectify the packet reordering problem by simply buffering and re-sequencing the out-of-order packets at the switch outputs. Our solution is based on the following observation of the queuing dynamics in the basic LBS: although two "back-to-back" packet arrivals A (earlier) and B (later) belonging to the same *switch flow* (i.e., they share the same input and output) may reach the output port out of order, the amount of reordering (i.e., how much earlier B arrives at the output port before A) is upper-bounded with high probability by a fairly small value. In particular, we formally bound the worst-case amount of time that a packet has to wait in these output reordering buffers before it is guaranteed to be ready for in-order departure with high probability, and we prove that this bound is *linear* with respect to the switch size. This linear bound is significant because previous approaches can add quadratic or cubic delays to the load-balanced switch. In addition, we use a hash-grouping method that further reduces resequencing delays significantly.

The second approach we considered is based on randomization. To prevent harmful effects in TCP performance due to out-of-order packets, only packets belonging to the same application flow (e.g. a TCP/IP flow) have to depart from their output port in order. Randomized load-balancing of application flows by means of hashing on the packet header is a well-known simple solution to this packet reordering problem in which all packets belonging to the same application flow are routed through the same intermediate port and hence the same path through the switch. Unfortunately, this method of load-balancing can lead to instability, depending on the mix of flow sizes and durations in the group of flows that gets randomly assigned to route through the same intermediate port. In our study, we show that the randomized load-balancing of application flows can be enhanced to provably guarantee both stability and packet ordering by extending the approach with *safety* mechanisms that can *uniformly diffuse* packets across the switch whenever there is a build-up of packets waiting to route through the some intermediate port.

1.3 Organization of the Dissertation

The rest of the dissertation is organized as follows. Chapter 2 provides a literature survey for the related research work; Chapters 3 and 4 describe our packet re-sequencing and TCP hashing approaches in details, before we conclude the dissertation in Chapter 5.

Besides the load-balanced switches, this dissertation also includes some theoretical results on the stability of a broad class of single-stage crossbar switching algorithms, which are attached in Appendix C.

CHAPTER 2
LITERATURE REVIEW

The idea of routing to a random intermediate port can be traced as far back as to the Valiant load-balancing (VLB) [9, 10] in the early 1980s. It was rediscovered by Chang et al. [1, 2] for designing load-balanced switches to mitigate routers' scaling challenges. In this chapter, we first provide a brief overview of the Valiant load-balancing techniques in Section 2.1, followed by a more detailed description of the load-balanced switch (LBS) architectures (Section 2.2), the existing solutions to the packet reordering problem in load-balanced switches (Section 2.3) and some other related works on the network-level traffic load balancing problems in hierarchical switch architectures (Section 2.4).

2.1 Valiant Load-Balancing and Its Applications

The scheme of routing through a randomly picked intermediate node en route to a packet's destination was first proposed by L. G. Valiant in his seminal work [9]. He showed that in an N-node binary cube network, the distributed two-hop randomized routing algorithm can route every packet to its destination within $O(\log N)$ time with overwhelming probability. Such randomized routing, which is often referred to as *Valiant load-balancing (VLB)* or *two-phase routing*, has the advantage of being decentralized, scalable and agnostic to the traffic load matrix. Since then, VLB has been widely used to improve the peformance of interconnection networks, such as reducing transmission delay [11], obtaining relief of adverse source-destination traffic patterns [12], and providing worst-case performance guarantees [13, 14]. It is also used for building scalable network switches [1, 2, 15, 16, 17, 18], for designing efficient and robust backbone networks [19, 20, 21, 22] and optical networks [23, 24, 25, 26], and for scaling data center networks [27, 28]. In particular, it was rediscovered by Chang et al. for designing load-balanced router switch fabrics [1]

to mitigate routers' scaling challenges, because it was difficult for conventional switch architectures, like centrally-scheduled input-queued crossbar switches, to keep up with the ever increasing traffic demand and link speed.

2.2 Load-Balanced Switch Architecture

As mentioned above, *load-balanced switch* (LBS) is a promising scalable class of switch architecture build upon the idea of Valiant load-balancing. As described in Chapter 1, they rely on two switching stages for routing packets. The first switching stage connects the first stage of input ports to the center stage of intermediate ports, and the second switching stage connects the center stage of intermediate ports to the final stage of output ports. Both switching stages execute a deterministic connection pattern such that each input is connected to each output of a switching stage at $1/N$th of the time. More specifically, the first switching fabric executes a periodic "increasing" sequence, that is, at any time slot t, each input port i is connected to the intermediate port $((i + t) \mod N) + 1$. The second switching fabric, on the other hand, executes a periodic "decreasing" sequence, that is, at any time slot t, each intermediate port m is connected to the output port $((m - t) \mod N) + 1$. Following the sequence of connection patterns in the first switching fabric, each input port i can "stripe" a frame of N packets, or all packets that are ready for service if there are less than N of them, to intermediate ports $1, 2, \ldots, N$ respectively, in N consecutive time slots. As explained before, two packets belonging to the same VOQ can go to two different intermediate ports, experience different queueing delays, and arrive at the output port out of order.

Compared with the traditional two-phase randomized routing scheme, it was shown that splitting traffic in a round-robin fashion has the same effect on link load as random splitting [1], although in theory, some very mild conditions, such as weakly mixing, need to be imposed on the arrival processes to preclude the pathological cases in which the destination output ports of packets arriving at an input port are perfectly synchronized with

their "sequence numbers modulo N" that create a severe load-imbalance across the intermediate ports. LBSes are much more scalable, in terms of both switch size and link speed, than conventional switch architectures. This is because, in LBS architectures, the connection pattern during every switching cycle at each switching stage is fixed and requires zero computation, and each port can forward packets in a fully distributed manner based only on local information.

2.3 Packet Reordering Problem in LBSes and the Existing Solutions

As mentioned in Chapter 1, although the basic LBS originally proposed in [1] is highly scalable, it has the critical problem that packets can depart out-of-order from the switch. In the basic LBS, consecutive packets at an input port are spread to all N intermediate ports upon arrival. These packets, going through different intermediate ports, may encounter different queuing delays. Therefore, a number of researchers have subsequently explored this packet ordering problem. This section briefly reviews existing solutions to the packet reordering problem in load-balanced switches, which can be grouped in accordance to their approach.

2.3.1 Randomization-Based

As mentioned in Chapter 1, packets belonging to the same application flow can be guaranteed to depart from their output port in order if they are forced to go through the same intermediate port. The selection of intermediate node can be easily achieved by hashing on the header field of every packet (source and destination IP addresses, source and destination ports, and protocol identification) to obtain a value from 1 to N. Despite its simplicity, the main drawback of this flow randomization approach is that stability cannot be guaranteed.

2.3.2 Aggregation-Based

An alternative class of algorithms to hashing is based on aggregation of packets into frames. One approach called Uniform Frame Spreading (UFS) [5] prevents reordering by requiring that each input first accumulates a full-frame of N packets, all going to the same output, before uniformly spreading the N packets to the N intermediate ports. Packets are accumulated in separate virtual output queues (VOQs) at each input for storing packets in accordance to their output. When a full-frame is available, the N packets are spread by placing one packet at each of the N intermediate ports. This ensures that the lengths of the queues of packets destined to the same output are the same at every intermediate port, which ensures every packet going to the same output experiences the same queuing delay independent of the path that it takes from input to output. Although it has been shown in [5] that UFS achieves 100% throughput for any admissible traffic pattern, the main drawback of UFS is that it suffers from long delays, $O(N^3)$ delay in the worst-case, due to the need to wait for a full-frame before transmission. The performance of UFS is particularly bad at light loads because slow packet arrivals lead to much longer accumulation times.

An alternative aggregation-based algorithm that avoids the need to wait for a full-frame is called Full Ordered Frames First (FOFF) [5]. As with UFS, FOFF maintains VOQs at each input. Whenever possible, FOFF will serve full-frames first. When there is no full-frame available, FOFF will serve the other queues in a round-robin manner. However, when incomplete frames are served, packets can arrive at the output out of order. It has been shown in [5] that the amount of reordering is always bounded by $O(N^2)$ with FOFF. Therefore, FOFF adds a reordering buffer of size $O(N^2)$ at each output to ensure that packets depart in order. It has been shown in [5] that FOFF achieves 100% throughput for any admissible traffic pattern, but the added reordering buffers lead to an $O(N^2)$ in packet delays.

Another aggregation-based algorithm called Padded Frames (PF) [4] was proposed to avoid the need to accumulate full-frames. Like FOFF, whenever possible, FOFF will serve

full-frames first. When no full-frame is available, PF will search among its VOQ at each input to find the longest one. If the length of the longest queue exceeds some threshold T, PF will pad the frame with fake packets to create a full-frame. This full-frame of packets, including the fake packets, are uniformly spread across the N intermediate ports, just like UFS. It has been shown in [4] that PF achieves 100% throughput for any admissible traffic pattern, but its worst-case delay bound is still $O(N^3)$.

Recently, an approach called Sprinklers [3] was proposed based on the idea of variable-size striping. Instead of requiring the accumulation of a full-frame of N packets before uniformly spreading the packets to all intermediate ports, Sprinklers uses the arrival rate of packets to a VOQ to determine a variable stripe size L. It then only requires the accumulation of L packets in a VOQ before uniformly spreading the L packets across a randomly chosen continuous block of L intermediate ports, where VOQs with slower arrival rates are given smaller stripe sizes. The approach can be seen as a hybrid between aggregation and randomization.

2.3.3 Matching-Based

Finally, packet ordering can be guaranteed in load-balanced switches via another approach called a Concurrent Matching Switch (CMS) [6]. Like hashing-based and aggregation-based load-balanced switch designs, CMS is also a fully distributed solution. However, instead of bounding the amount of packet reordering through the switch, or requiring packet aggregation, a CMS enforces packet ordering throughout the switch by using a fully distributed load-balanced scheduling approach. Instead of load-balancing packets, a CMS load-balances request tokens among intermediate ports, where each intermediate port concurrently solves a local matching problem based only on its local token count. Then, each intermediate port independently selects a VOQ from each input to serve, such that the packets selected can traverse the two load-balanced switch stages without conflicts. Packets from selected VOQs depart in order from the inputs, through the intermediate ports,

and finally through the outputs. Each intermediate port has N time slots to perform each matching, so the complexity of existing matching algorithms can be amortized by a factor of N.

2.4 Load-Balancing in Hierarchical Switch Architectures

Besides existing works on LBS architectures, recently Dixit et al. [29] studied empirically the effects of random load-balancing on packet ordering in symmetric data center networks such as multi-rooted tree topologies. Their empirical results confirm our observations that the amount of packet reordering that can occur is quite limited. We believe that our analysis framework can be applied to formally bound the amount of packet reordering in that data center setting as well, and we plan to provide analytical bounds for symmetric data center networks in future work.

In addition, the technique of flow hashing (i.e., "TCP-hashing") has also been used to solve network-level traffic load balancing problems in hierarchical switch architectures (e.g., a data center) [30, 28]. It is widely known [31, 32, 33, 34, 35, 36] however that a naive flow hashing scheme (e.g., Equal-Cost Multipath) does not perform well in balancing loads since it could map too many large long-lived flows to the same path, leading to instability. One possible way of avoiding this instability issue is to divide each flow into small pieces and route them along different paths [37, 38], but this may again lead to the packet reorder problem. To address this packet reorder problem, in CONGA [37], packets in a flow are divided into bursts of packets (with a sizable time gap between any two consecutive bursts) called "flowlets"; if the time gap between two consecutive flowlets is larger than the maximum difference in latency among the paths, the second flowlet can be sent along a different path than the first without causing packet reordering. However, there could be a long tail in the length distribution of the flowlets, and in this case the collision on the hashing of elephant flowlets can still result in non-negligible congestion issue [38]. Presto [38] avoids this congestion issue by dividing each flow into uniform-sized pieces

called flowcells and re-sequencing the out-of-order packets at the receiver, with a caveat that a significantly out-of-order (late) packet might be dropped due to the timeout of the re-sequencing timer.

CHAPTER 3

RS-LBS: A SIMPLE RE-SEQUENCING LOAD-BALANCED SWITCH BASED ON ANALYTICAL PACKET REORDERING BOUNDS

3.1 Overview

In this chapter, we propose a natural solution to rectify packet reordering problem in LBS, named the RS-LBS (Re-Sequencing Load-Balanced Switch), that incurs significantly lower packet delays compared to the aforementioned packet-aggregation-based solutions. Our solution is based on the following observation of the queuing dynamics in the basic LBS: although two "back-to-back" packet arrivals A (earlier) and B (later) belonging to the same *switch flow* (i.e., they share the same input and output) may reach the output port out of order as explained above, the amount of reordering (i.e., how much earlier B arrives at the output port before A) is upper-bounded with high probability by a fairly small value. The reason for this is that under the round-robin load dispatching mechanism of the basic LBS, the two VOQs at two different intermediate ports that A and B traverse through respectively, have almost independent and stochastically identical queuing processes. Hence the delays A and B experience at their respective intermediate ports are almost i.i.d. (independent and identically distributed) random variables, so with high probability the latter cannot be much larger than the former.

Our basic RS-LBS scheme is for an output port to hold each out-of-order packet, for a period of time not exceeding this (small) upper-bound, for re-sequencing. While packets that are severely out-of-order, i.e., those causing this upper-bound to be exceeded, may remain out-of-order after the re-sequencing, they represent a tiny percentage of the network traffic and may negatively impact the performance of an even tinier percentage of TCP flows. We prove that this upper bound is *linear* with respect to the switch size, i.e., $O(N)$,

under any traffic arrival process for which the switch is stable. Hence with a worst-case delay of at most $O(N)$ due to this re-sequencing, we can ensure that an overwhelming majority of packets exit the switch in-order; in fact, since the output port holds a packet for only as long as is necessary, the average delay of packets is much smaller than this worst-case delay, as will be shown in Section 3.5. In contrast, all other present solutions to the packet reordering problem incur $O(N^2)$ or $O(N^3)$ delays.

Our solution can be further refined to reduce the average delay of packets by relaxing the packet ordering semantics as follows. The semantics discussed above, which we refer to as *per-switch-flow ordering*, aims to ensure that packets arriving to the same input port departs from the corresponding output port in the same order. The per-switch-flow ordering is unnecessarily conservative because if two out-of-order packets within the same switch flow belong to *two different TCP flows*, then having them depart the switch in a different order would not negatively impact either TCP flow. The most relaxed yet still harmless semantics would be to only ensure the sequencing of packets within a TCP flow, which we refer to as *per-TCP-flow ordering*. With per-TCP-flow ordering, the average packet delay (due to re-sequencing) can be reduced significantly, as an out-of-order packet no longer has to be held waiting for packets from other TCP flows. This relaxed semantics, however, is computationally expensive to implement, since to do so would require each output port to perform per-TCP-flow queuing and re-sequencing. We propose the use of a slightly less relaxed semantics called *per-hashed-group ordering*. With per-hashed-group ordering, incoming packets at an input port with the same departure output port (i.e., the same switch flow) are hashed to one of K counters and are numbered by the corresponding hashed counter. Per-hashed-group ordering simply requires two packets belonging to the same *hashed group* to depart in order in accordance to their assigned sequence numbers[1]. This slightly less relaxed semantics results in almost the same level of reduction to the average delay as per-TCP-flow ordering, but yet it has an implementation complexity comparable

[1] In contrast, with per-switch-flow ordering, incoming packets are numbered in accordance to their arrivals to the same switch flow.

to that of per-switch-flow ordering.

The rest of the chapter is organized as follows. In Section 3.2, we analyze the amount of packet reordering under in a basic LBS. In Sections 3.3 and 3.4, we present the basic and enhanced (with hash-grouping) RS-LBS schemes. In Section 3.5, we compare the average delay performance of our re-sequencing schemes with existing LBS architectures. Finally, we conclude the chapter in Section 3.6

3.2 Analysis of Packet Reordering Probability

As explained earlier, the efficacy of our natural solution hinges upon the premise that only a tiny percentage of packet pairs are severely reordered when they reach their output ports. A packet pair A and B within the same switch flow, A arriving earlier at the input port than B, are considered severely reordered, if B arrives earlier than A by at least a lateness threshold θ. Clearly, the larger is this lateness threshold θ, the smaller is the proportion of severely reordered packet pairs.

In this section, we show, through a careful analysis, that even with a fairly small lateness threshold of $O(N)$, the proportion of severely reordered packet pairs can go down to a tiny number. Note this $O(N)$ result holds for any traffic arrival process for which the switch is stable. However, in analyzing the constant factor (e.g., "23" in the following example) in this $O(N)$, we assume the arrival process is Poisson. For example, we will show that when the switch is 90% loaded under Poisson traffic and the lateness threshold is set to $23N$, no more than 1 out of 1,000 "back-to-back" packet (arrival) pairs in the same switch flow will be severely reordered (There is a quotation mark around the word "back-to-back" here, since between this pair of packet arrivals there can be other packet arrivals belonging to other switch flows to this input port). This property allows our natural solution of re-sequencing packets at the output ports to remove all but a tiny proportion of out-of-order packets while introducing at most $O(N)$ re-sequencing delay to each packet.

3.2.1 Problem formulation

Consider two "back-to-back" packets A and B *in the same swich flow* where A arrives before B. Let m_1 and m_2 be the intermediate ports that A and B transit through respectively, and j be their common destination output port. Then A and B both transmit through the j-th VOQ (for buffering all packets destined for output port j) at the corresponding intermediate ports. If these two packets happen to transit through the same intermediate port, i.e., $m_1 = m_2$, then they clearly will not be reordered because they go through the same intermediate VOQ. Hence we assume $m_1 \neq m_2$ when we analyze the reordering probability.

Let L_1 and L_2 be random variables denoting the queuing delays that A and B experience, at the j-th VOQs of the intermediate ports m_1 and m_2, respectively. Define a *cycle* as N *time slots*. Even if A and B depart from the input port simultaneously, B can arrive at the output port at most $L_1 - L_2$ cycles before A. Based on this observation, given any lateness threshold $\theta > 0$ (in the unit of cycles), we aim to derive $P(L_1 - L_2 > \theta)$, the probability that this pair of "back-to-back" packets is severely reordered (with respect to θ).

Note this measure of packet reordering, namely *"back-to-back" packet reordering probability*, is different than the conventional measure readers may have in mind. In particular, this measure does not account for all packet reordering cases that may impact TCP performance, which would be accounted for in the conventional measure. For example, consider 5 "consecutive" packets A, B, C, D, and E in the same switch flow. This measure accounts only for the reorderings between 4 pairs of "back-to-back" packets: (A, B), (B, C), (C, D) and (D, E), but not for other pairs such as (A, D) and (C, E).

We emphasize, however, that our measure is only used for the theoretical analysis, including the $O(N)$ packet waiting time proof, and for deciding on the right parameters used in our packet buffering and re-sequencing schemes. We will use the conventional measure of packet reordering when evaluating the efficacy of our schemes. We further emphasize

that our $O(N)$ proof (established below) remains valid according to the conventional measure, because our measure turns out to be more conservative than the conventional measure for the following two reasons.

First, in estimating the reordering probability of a pair of "back-to-back" packets, we assume these two packets depart from the input port, to their respective intermediate ports, simultaneously. However, in reality, two "back-to-back" packet arrivals (say A and B) within the same switch flow can be separated by τ packet arrivals, where τ can be tens, hundreds or more packets from other switch flows. These τ intervening packets from other switch flows would cause a gap of at least τ time slots (or τ/N cycles) between the departure times of A and B from the input port. The reordering probability $P(L_1 - L_2 > \theta + \tau/N)$ can be much smaller than $P(L_1 - L_2 > \theta)$, when τ is tens, hundreds or more. Second, the gap between the departure times of other pairs of packets within the same switch flow, say A and C, is typically much wider than this τ, the gap between the departure times of "back-to-back" pairs, hence the reordering probability between any such pair is generally negligible compared to that between a "back-to-back" pair.

As mentioned earlier, each input port distributes incoming packets, regardless of their destination output ports (i.e., the switch flows they belong to), uniformly to all N intermediate ports in a round-robin fashion. It has been shown in [1] that, under very mild assumptions on the traffic arrival process to the switch (i.e., to its input ports) such as weakly mixing, for any j, m_1, and m_2, the packet arrival process to the j-th VOQ of the intermediate port m_1 and that to the j-th VOQ of the intermediate port m_2 can be viewed as i.i.d. stochastic random processes. Since these two intermediate VOQs also have the same deterministic service process, more specifically switching one packet to output port j every N time slots, the queuing processes of VOQs are i.i.d. Hence the queuing delays L_1 and L_2 are i.i.d. random variables.

Let the packet arrival process to each of the N^2 switch flows be independent of each other and be stationary. Let $\lambda_{ij} \geq 0$ be the average arrival rate of packets arriving at input

port i destined for output port j. Define $\lambda_j \equiv \sum_{i=1}^{N} \lambda_{ij}$ to be the total average arrival rate for output port j. Let the service rate of each input or intermediate port be 1. For this arrival process (to the switch) to be admissible, we must have

$$\sum_{j=1}^{N} \lambda_{ij} < 1, \qquad i = 1, 2, \ldots, N,$$
$$\sum_{i=1}^{N} \lambda_{ij} < 1, \qquad j = 1, 2, \ldots, N.$$

3.2.2 The $O(N)$ Proof

We again emphasize that Poisson assumption is not needed for deriving this $O(N)$ proof. Define $\pi_n \equiv P(L_1 = n)$ for $n = 0, 1, 2, \ldots$. Recall L_1 is the queueing delay packet A experiences at the j-th VOQ of the intermediate port m_1. Since the arrival process to this VOQ is stationary, as stated before, and the service process is deterministic (and hence stationary), the distribution of L_1 is precisely the stationary queueing delay distribution of this VOQ. Since the term N does not appear in either the arrival rate λ_j or the service rate 1 (i.e., 1 packet every cycle or N time slots), it should not appear in the distribution of L_1 (i.e., any of the π_n terms). Hence each π_n term is a function of only λ and n. Since L_1 and L_2 are i.i.d., by the convolution formula, we have,

$$\begin{aligned} P(L_1 - L_2 \geq \theta) &= \sum_{k=\theta}^{\infty} P(L_1 - L_2 = k) \\ &= \sum_{k=\theta}^{\infty} \sum_{i=0}^{\infty} P(L_2 = i) P(L_1 = i + k) \\ &= \sum_{k=\theta}^{\infty} \sum_{i=0}^{\infty} \pi_i \pi_{i+k}. \end{aligned}$$

Note that the term N does not appear in the above formula. This implies that $P(L_1 - L_2 \geq \theta)$ is a function of only θ and λ_j. Hence given a load factor λ_j and a target $P(L_1 - L_2 \geq \theta)$ value (say 10^{-3}), the θ value that allows us to reach (i.e., go under) this target

$P(L_1 - L_2 \geq \theta)$ value is a constant (with respect to N). Note that under any admissible traffic arrival process to the switch $P(L_1 < \infty) = 1$, so any nonnegative target $P(L_1 - L_2 \geq \theta)$ value can be reached, no matter how small it is. Hence the output needs to hold packet B for up to θ cycles, or θN time slots, to achieve this target $P(L_1 - L_2 \geq \theta)$ value. *This proves that at most $O(N)$ amount of buffering (and waiting) is needed at each output port to re-sequence the vast majority of packets.*

3.2.3 Analysis of the Constant Factor θ

In analyzing the constant factor θ in this $O(N)$ result, we assume that arrival processes to all N^2 switch flows are Poisson. With this Poisson assumption, the queuing process of the j-th intermediate VOQ at intermediate port m_1 can be viewed as M/D/1 (i.e., Poisson arrival[2] and deterministic service time). Then the distribution of L_1 is equal to the stationary queueing delay distribution of this M/D/1 queue. The same can be said about the queuing process of the j-th intermediate VOQ at intermediate m_2, and about L_2.

The distribution of L_1 and its numerical computation

Recall that the (normalized) arrival rate to this M/D/1 queue (i.e., the j-th VOQ at intermediate port m_1) is λ_j and its service rate is 1 (i.e., 1 packet every cycle or N time slots). For the ease and clarity of the presentation, we drop the subscript j from λ_j and denote the total load to the output port j simply as λ. By the Pollaczek-Khinchin formula [39], we have

$$\pi_0 = 1 - \lambda \tag{3.1}$$

$$\pi_1 = (1 - \lambda)(e^{\lambda} - 1) \tag{3.2}$$

$$\pi_n = (1 - \lambda) \sum_{k=1}^{n-1} e^{k\lambda} (-1)^{n-k} \left[\frac{(k\lambda)^{n-k}}{(n-k)!} + \frac{(k\lambda)^{n-k-1}}{(n-k-1)!} \right]$$

[2]Due to the discretization of time into slots by the switch, this arrival process (to an intermediate VOQ) is i.i.d. Bernoulli, not Poisson, even when the arrival process to the input port is assumed to be Poisson. However, when N is large, which is what LBS is designed for, the Bernoulli process is stochastically very close to the Poisson process.

$$+ (1 - \lambda)e^{n\lambda}. \tag{3.3}$$

Unfortunately, computing (3.3) leads to numerical instability, since it is a summation of alternating sign large absolute values. Our tests show that numerical computation via Matlab can accurately compute only up to π_{10} when $\lambda = 0.1$ (10% loaded) and up to π_{20} when $\lambda = 0.9$ (90% loaded). We overcome the problem by using following upper bound of π_n that is numerically stable [40]:

$$\pi_n \leq -\frac{\alpha_0}{\zeta_0^{n+1}} + \frac{M_1(r)}{r^n}, \;\; n = 0, 1, 2, \ldots \tag{3.4}$$

where ζ_0, α_0, r and $M_1(r)$ are parameters that arise in analyzing this distribution via complex analysis.

Fig. 3.1 presents the numerical results of this upper bound for load factors $\lambda = 0.1, 0.5, 0.9$, with the parameters listed in Table 3.1. We see from the figure that the upper bound given by (3.4) is quite tight when n is large. Thus whenever n is large enough so that π_n cannot be computed accurately using equation (3.3), it can be tightly bounded using equation (3.4).

Table 3.1: Parameters for the upper bound in equation (3.4) (taken from [40] with a correction).

λ	ζ_0	α_0	r	$M_1(r)$
0.1	37.1	-444.8	84.0	384.0
0.5	3.5	-5.8	16.0	60.6
0.9	1.2	-0.26	8.0	0.95

Piecing Everything Together

Combining Equations (3.1), (3.2), (3.3) and (3.4), we obtain the following upper bound of $P(L_1 - L_2 \geq \theta)$, which is also a close approximation of it. Here d is the aforementioned threshold such that, when $n > d$, an accurate numerical computation of π_n using equation (3.3) becomes impossible. Recall that, as explained before, when $n > d$, the π_n terms in

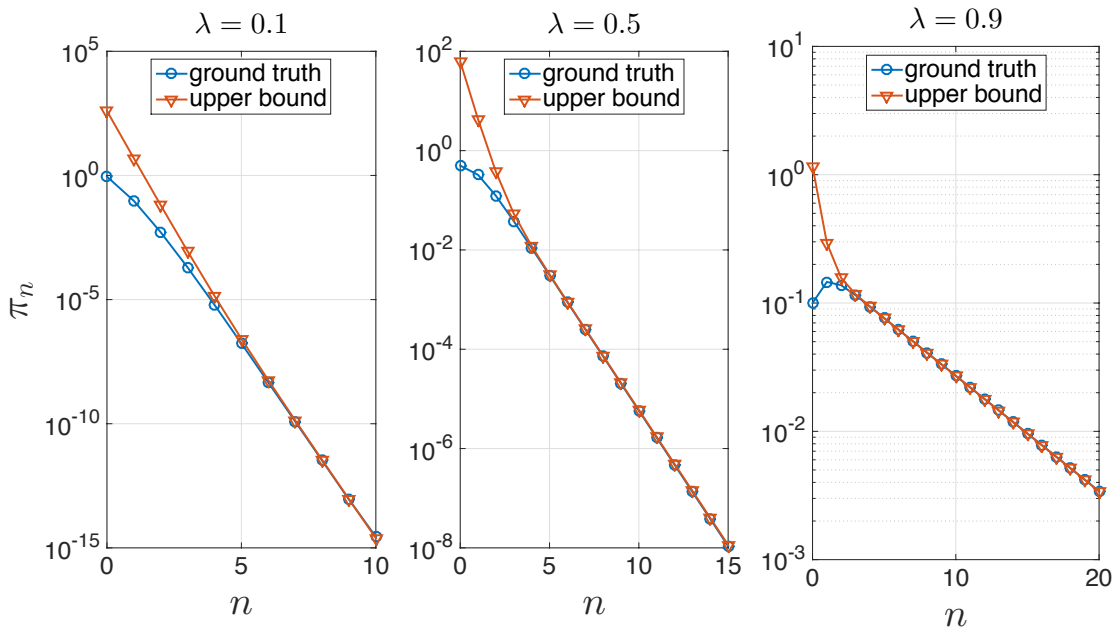


Figure 3.1: Comparing the true value of π_n (denoted *ground truth*) with its upper bound under $\lambda = 0.1, 0.5, 0.9$.

equation (3.3) can each be tightly bounded using equation (3.4), and their sums result in the right-hand-side terms in inequalities (3.5) and (3.6). The proof of Theorem 1 is provided in Appendix A.1.

Theorem 1. *When $\theta > d$, we have*

$$P(L_1 - L_2 \geq \theta) \leq C_\zeta^{(d)} \frac{1}{\zeta_0^\theta} + C_r^{(d)} \frac{1}{r^\theta} \tag{3.5}$$

When $0 \leq \theta \leq d$, we have

$$P(L_1 - L_2 \geq \theta) \leq \sum_{k=\theta}^{d} \sum_{i=0}^{d-k} \pi_i \pi_{i+k} + C_\zeta^{(d)} \frac{1}{\zeta_0^{d+1}} + C_r^{(d)} \frac{1}{r^{d+1}} + \widetilde{C}_\zeta^{(\theta,d)} + \widetilde{C}_r^{(\theta,d)}, \tag{3.6}$$

where

$$C_\zeta^{(d)} = \left(-\sum_{i=0}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{i+1}} + \frac{\alpha_0^2}{\zeta_0^{2d+4}(1-\zeta_0^{-2})} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1} r^{-1})} \right) \frac{1}{(1-\zeta_0^{-1})}$$

$$C_r^{(d)} = \left(\sum_{i=0}^{d} \frac{\pi_i M_1(r)}{r^i} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1} r^{-1})} + \frac{M_1^2(r)}{r^{2d+2}(1-r^{-2})} \right) \frac{1}{(1-r^{-1})}$$

$$\widetilde{C}_{\zeta}^{(\theta,d)} = \left(\frac{\alpha_0^2}{\zeta_0^{2d+4}(1-\zeta_0^{-2})} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1}r^{-1})} \right) \sum_{k=\theta}^{d} \frac{1}{\zeta_0^k} - \sum_{k=\theta}^{d} \sum_{i=d-k+1}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{k+i+1}}$$

$$\widetilde{C}_{r}^{(\theta,d)} = - \left(\frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1}r^{-1})} - \frac{M_1^2(r)}{r^{2d+2}(1-r^{-2})} \right) \sum_{k=\theta}^{d} \frac{1}{r^k} + \sum_{k=\theta}^{d} \sum_{i=d-k+1}^{d} \frac{\pi_i M_1(r)}{r^{k+i}}.$$

Numerical Results of $P(L_1 - L_2 \geq \theta)$

Let $\overline{P}(\theta)$ denote the tight upper bound of $P(L_1 - L_2 \geq \theta)$ given in the right hand sides of (3.5) and (3.6). $\overline{P}(\theta)$ for $\lambda = 0.1, 0.5, 0.9$ is plotted in Figure 3.2. From their definitions given in [40], we can infer that $r \geq \zeta_0 > 1$, $\alpha_0 < 0$ and $M_1(r) > 0$, and thus $\overline{P}(\theta)$ is a decreasing function of θ. This monotonicity can also be seen from Figure 3.2.

Given the monotonicity of $\overline{P}(\theta)$, we can define L_ϵ as follows. For any $\epsilon > 0$, let L_ϵ be the smallest nonnegative number such that $\overline{P}(\theta) \leq \epsilon$ if and only if $\theta \geq L_\epsilon$. Since the service rate of an intermediate port VOQ is $1/N$, as explained earlier, with probability no more than ϵ, packet B arrives at output j at least $L_\epsilon N$ time slots earlier than A. So if B waits up to $L_\epsilon N$ time slots at the output port, the probability that B departs from the output port earlier than A should be no more than ϵ. In other words, if we maintain a re-sequencing buffer of size $\lceil L_\epsilon \rceil N$ at each output port, we can expect the packet reordering probability to be no more than ϵ. For example, for $\epsilon = 10^{-3}$ and $\lambda = 0.9$, we have $\lceil L_\epsilon \rceil = 23$. In other words, if each output port maintains a re-sequencing buffer of size $23N$, the switch can keep the packet reordering probability below 10^{-3} when the traffic load is no more than 90%.

3.3 The Basic RS-LBS Scheme

In the previous section, we proved that a "back-to-back" packet pair A and B in the same switch flow is, with high probability, re-ordered by at most $O(N)$ time slots. Hence, the core idea of our basic RS-LBS scheme is, for each output port j, to hold out-of-order packets for at most $O(N)$ time slots and re-sequence them to the extent possible under this

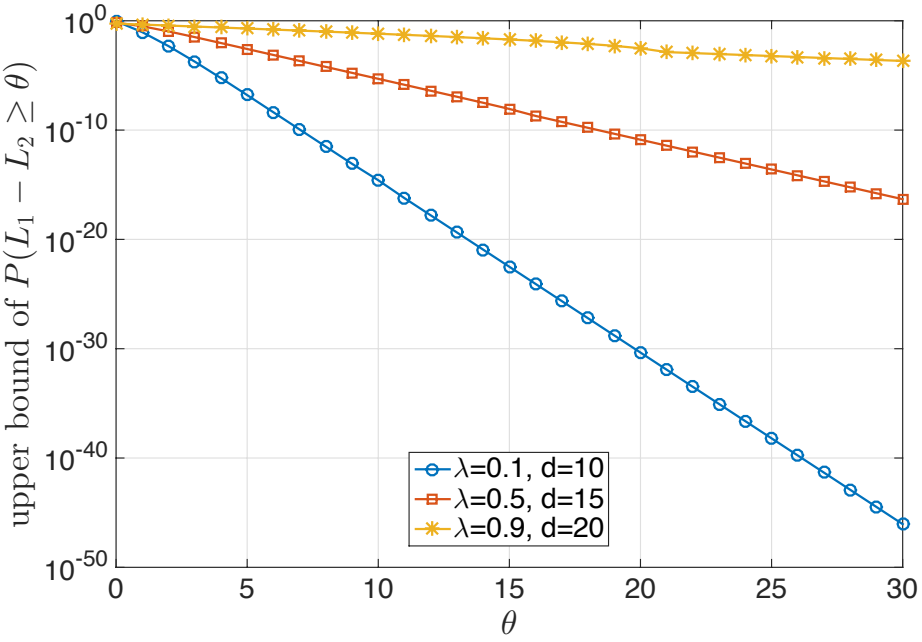


Figure 3.2: $\overline{P}(\theta)$, the upper bound of $P(L_1 - L_2 \geq \theta)$.

$O(N)$ maximum holding time constraint. With this idea, in theory at most $P(L_1 - L_2 \geq \theta)$ fraction of packets will remain re-ordered after this buffering and re-sequencing.

3.3.1 Re-sequencing Policies

The key idea of the basic RS-LBS scheme is, however, insufficiently detailed to define our scheme. The following packet re-sequencing policy at each output port fills the gap:

- Whenever a packet has waited for θN (equal to buffer size) time slots at an output port buffer, we mark this packet (and all packets before it) as expired.
- We mark a packet as in-order, if and when all packets have departed from the output port that are in the same switch flow as, and arrived earlier at the input port than, the packet.
- A packet can depart from the output port whenever it is in-order or expired. If there are multiple expired packets in the buffer, they will be served in the right order (according to the order they arrive at the input port as indicated by an internal sequence number field assigned by the input port).

Since the policy guarantees that no packet waits more than θN time slots in the buffer, the re-sequencing delay is no larger than θN. A drawback of this policy, however, is that it does not fully utilize the buffer space, because it may force an expired packet to depart from the output port *unnecessarily* (i.e., even when the buffer is not full). An alternative policy that fully utilizes the buffer is as follows:

- Each out-of-order packet shall keep waiting in the buffer until it becomes in-order or the buffer is full.
- When the buffer is full and there are no in-order packets in the buffer, we mark the packet that arrived at the switch (to any input port) the earliest as "expired".
- A packet can depart from the output port whenever it is in-order or expired.

The second policy essentially keeps a packet in the buffer for as long as possible in hopes of getting it back in-order. It is more aggressive than the first policy in reducing the proportion of reordered packets, so it should result in a smaller number of packet re-orderings than the first policy. However, conceivably there is a tradeoff here: the second policy may result in a larger average re-sequencing delay than the first policy. To make an informed decision as to which policy to adopt, we compared the reordering proportions and re-sequencing delays of these two policies via simulations. Here is a summary of the observations:

- The re-sequencing delay of first policy is slightly smaller than that of the second when the buffer size is small ($W = 2N$ or so), but are almost the same when the buffer size is large ($W \geq 5N$).

- The re-ordering probability of the second policy is much smaller than that of the first one.

We conclude from these observations that the second policy achieves a much better tradeoff between packet reordering probability and re-sequencing delay. In addition, as will be shown in Section 3.5.2, the re-sequencing delay will be even less an issue in the enhanced

(by hash-grouping) RS-LBS scheme. Hence, we adopt the second policy in both the basic and the enhanced RS-LBS schemes.

3.3.2 Implementation Issues

In this section, we explain some implementation issues regarding the aforementioned second packet re-sequencing policy that we adopt for our RS-LBS schemes. At any input port, an internal (switch-wide) header is added to each incoming packet that contains a timestamp, corresponding to the arrival time (slot) of the packet, and the identifier of the input port. Information contained in this header is used by the output ports for implementing the packet re-sequencing policy. At an output port, the packet re-sequencing operation dictated by this policy can be implemented using a heap of $O(N)$ nodes, and hence has a computational complexity of $O(\log N)$ per packet. This complexity can be further decreased to $O(1)$ using *pipelined* implementations of heaps or similar sorting data structures [41, 42, 43].

3.3.3 Stability of the Basic RS-LBS Scheme

Since our scheme holds at most $O(N)$ additional (waiting) packets in each output port buffer, compared to the basic LBS, our scheme is stable for any traffic arrival process for which the basic LBS is stable. Note that, for all practical purposes, we may consider the basic LBS to be stable under all aforementioned admissible traffic arrival processes, although in theory, some very mild conditions, such as weakly mixing, need to be imposed on the arrival processes to preclude the pathological cases in which the destination output ports of packets arriving at an input port are perfectly synchronized with their "sequence numbers modulo N" that create a severe load-imbalance across the intermediate ports. We refer readers to [1] for further details on the mild conditions and the pathological cases they aim to preclude.

3.4 Improving the performance by hash-grouping

As explained in the introduction, the only type of packet reordering that negatively impacts TCP performance is the reordering between two packets that belong to the same TCP flow. However, our basic RS-LBS scheme is oblivious to the flow identifier of a packet, so a packet already in-order within its own TCP flow may well be out-of-order within its switch flow, and be kept waiting in the output port buffer. Clearly, this wait time unnecessarily increases the average re-sequencing delay of a packet. A natural solution to this problem is to make each output port aware of the (TCP) flow identifier information in the packets. This solution, however, requires the output port to perform per-TCP-flow buffering and queuing, which is prohibitively expensive computationally.

We use a *hash-grouping* technique that was first introduced in [44], which significantly enhances the performance of the basic RS-LBS scheme, but yet has almost the same computational complexity and implementation cost as the basic scheme. The key idea is that, at each input port, the packets inside each switch flow are demultiplexed into K virtual groups via hashing. In other words, the (TCP) flow identifier of each packet is hashed to produce the index of the group to which the packet belongs. Note that since all packets of a TCP flow have the same flow identifier, they will be assigned to the same virtual group via hashing.

The packet re-sequencing policy needs only to be slightly revised to take advantage of hash-grouping. A packet in a hashed group is considered in-order if it is in-order within its own group, even though it is not in-order within its switch flow. With a sufficiently large number of groups, each group contains only a small number of active TCP flows at any given time. In this case, a large fraction of unnecessary wait time is avoided, and the resulting re-sequencing delay becomes so small that it is almost the same as if the output ports were to enforce correct packet ordering only within each TCP flow. We will demonstrate the efficacy of this technique in Section 3.5.2.

3.5 Evaluation

In this section, we compare the performance of our proposed approach with other existing load-balanced switching algorithms, including the basic LBS [1], Uniform Frame Spreading (UFS) [5], Full-Ordered Frame First (FOFF) [5], Padded Frames (PF) [4], and the recently proposed Sprinklers scheme [3]. Although the basic LBS (labeled "Basic" in subsequent figures) does not guarantee packet ordering, it is included in our comparisons because it provides a lower bound on the average delay that any LBS-based scheme can achieve. UFS, FOFF, PF, and Sprinklers are all known to provide fairly good performance while guaranteeing packet ordering.

Throughout this section, we assume $N = 64$. The normalized traffic load λ injected into the switch ranges from 0.1 to 0.95, while the output buffer size W varies from 0 to ∞ (i.e., no restriction on the buffer size). We also vary the traffic patterns in our evaluation. Our simulation covers 4 types of traffic matrices: uniform, quasi-diagonal, log-diagonal and diagonal. The load matrices are listed in the order of how skewed the traffic is to an output port: from uniform being the least skewed, to diagonal being the most skewed.

In order to simulate the effects of distributing TCP/UDP application flows into hashed groups, we generate application flows over the simulation period using flow statistics measured from a real-world Internet traffic trace. Specifically, we assume that the arrival of new application flows to an input port follows a Poisson process, and the rate and duration of each new application flow, viewed as a random vector, follows the corresponding joint empirical distribution measured from the traffic trace under consideration. The rate of this Poisson process is set according to the intended traffic rate λ of the input port in a simulation run, and the measured empirical average size (number of packets) of an applications flow.

The trace used was donated privately to the authors and was collected by University of North Carolina (UNC) on a 1 Gbps access link connecting the campus to the rest of the

Internet on April 24, 2003. It contains $198,944,706$ packet headers and around 13.5 million flows. In our simulation study, incoming traffic to each input port is generated according to the empirical flow statistics measured from this trace.

In the remainder of this section, we will first look at the performance of the basic RS-LBS scheme, followed by the performance of the enhanced RS-LBS scheme (with hash-grouping).

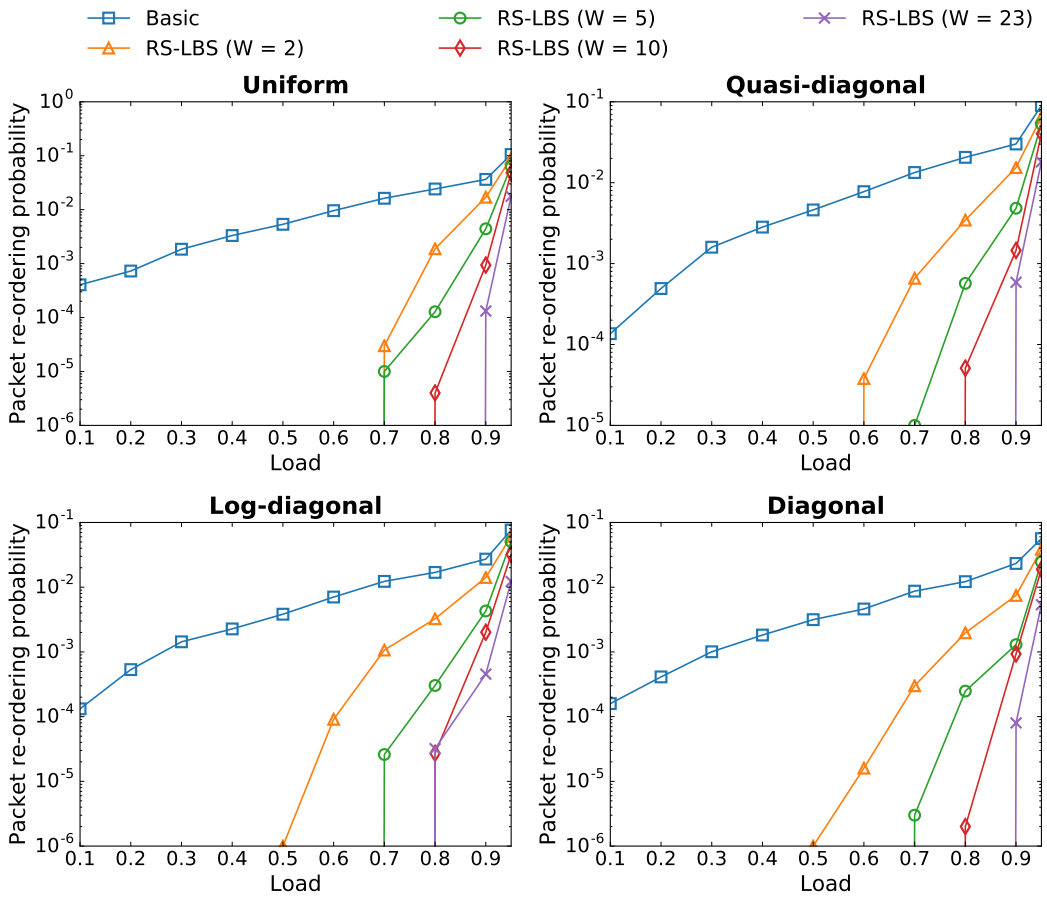


Figure 3.3: Packet reordering probability of the RS-LBS scheme, with various buffer sizes W.

3.5.1 Performance of Basic RS-LBS

The simulation results of the packet reordering probability and mean delay of our approach are shown in Fig. 3.3 and Fig. 3.4 respectively. As explained earlier, for evaluating em-

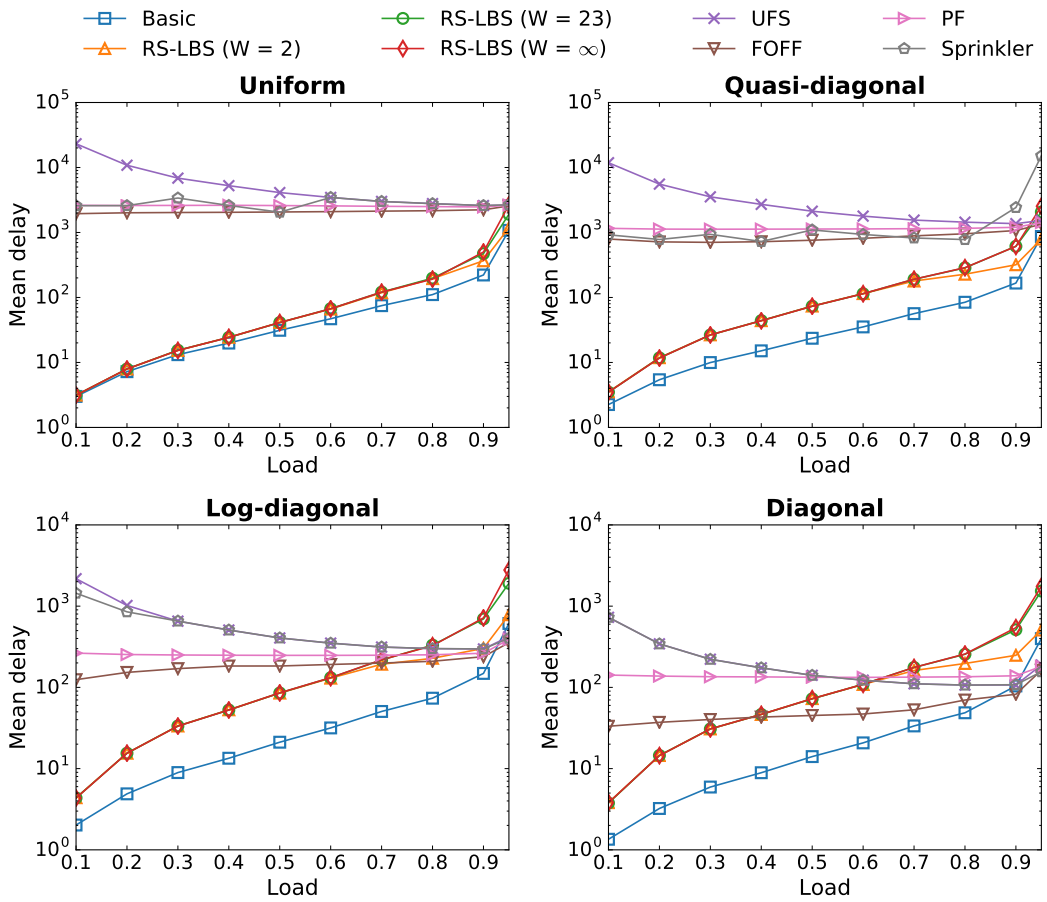


Figure 3.4: Average delay of the RS-LBS scheme, with various buffer sizes W, compared to UFS, FOFF, PF, and Sprinklers.

pirical performance we use the conventional measure of packet reordering: a packet A is considered re-ordered if and only if it departs from the output port earlier than another packet in the same TCP flow that arrives at the input port earlier than A. The infinite output buffer size case ("RS-LBS ($W = \infty$)") is when all out-of-order packets wait at the output port until they are in-order. This guarantees a zero packet reordering probability, and its delay can be taken as an upper bound for the re-sequencing delay introduced by our scheme.

Fig. 3.3 presents the packet reordering probability of our RS-LBS scheme. The packet reordering probabilities are in line with our analysis in Section 3.2. For instance, when the traffic load is $\lambda = 0.9$, for all 4 traffic patterns, the packet reordering probability is around

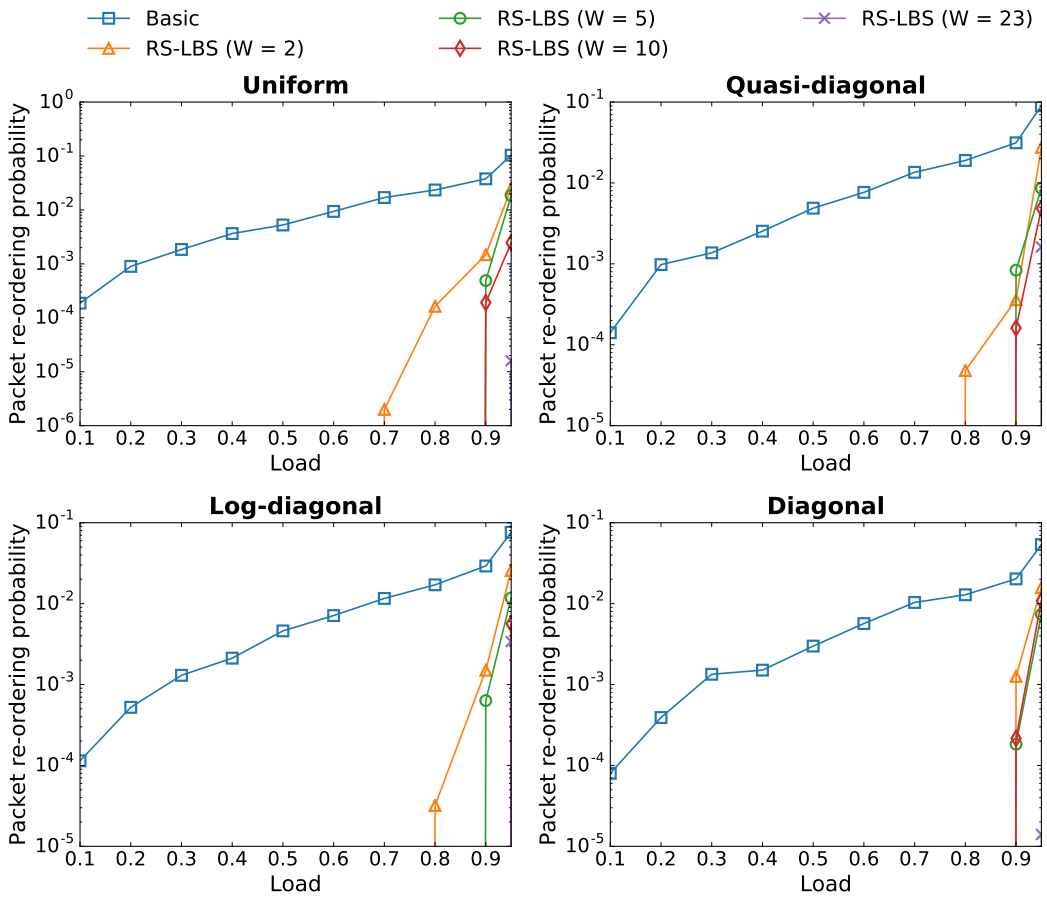


Figure 3.5: Packet reordering probability of the RS-LBS scheme with hash-grouping ($K = 1000$ groups).

10^{-3} when the buffer size $W = 10N$ and is around 10^{-4} when $W = 23N$. Interestingly, even with $W = 2$, the packet reordering probability is drastically reduced for light loads compared to the basic LBS.

Fig. 3.4 presents the average delay. Compared to the other LBS-based schemes, the delay of our RS-LBS scheme looks fairly good for uniform and quasi-diagonal traffic, but this is not the case for log-diagonal and diagonal traffic. We explain this observation as follows:

- The frame-based load-balancing algorithms, such as UFS, FOFF and PF, have very good delay performance for diagonal or log-diagonal traffic since aggregating a full frame when the traffic pattern is "concentrated" is relatively effortless.

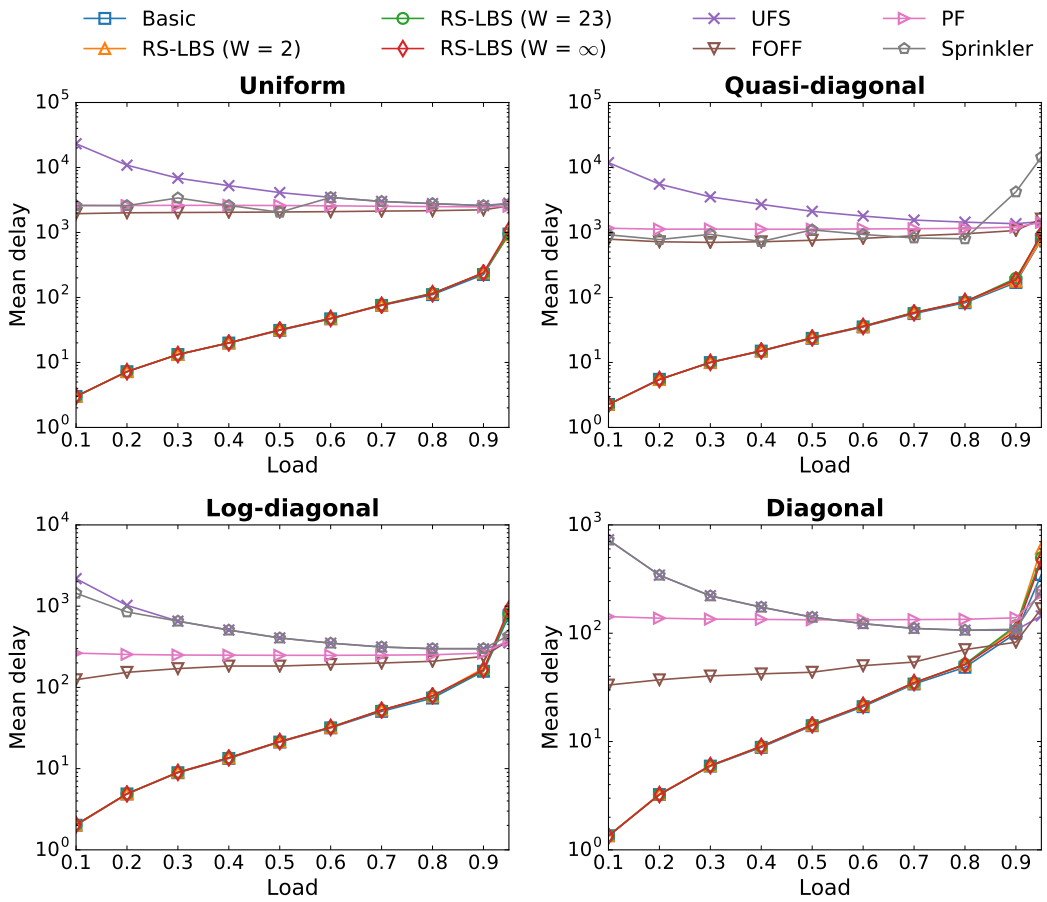


Figure 3.6: Average delay of the RS-LBS scheme with hash-grouping ($K = 1000$ groups), compared to UFS, FOFF, PF, and Sprinklers.

- In contrast, the load-balanced stage spreads traffic from the input ports uniformly across the intermediate ports, so a "concentrated" traffic pattern provides no benefit to the basic LBS and our scheme. The queue length distribution at the intermediate ports stay roughly the same no matter what the traffic pattern is (recall that no explicit assumption was made on the traffic matrix for the derivation of the bound on the packet reordering probability in Section 3.2.3), as does the re-sequencing delay.

In summary, the queuing delay of UFS, FOFF and PF is small for diagonal or log-diagonal traffic while the re-sequencing delay of RS-LBS remains unchanged, as concentrated traffic benefits the frame-based schemes. However, as we shall see below, the introduction of the "hash-grouping" technique (from Section 3.4) overcomes this weakness.

3.5.2 Performance of Enhanced (with Hash-Grouping) RS-LBS

As mentioned earlier, our natural solution can be enhanced by relaxing the packet reordering semantics and implementing the hash-grouping technique. In our simulations, the number of virtual groups for each switch flow is set to $K = 1,000$.

Fig. 3.5 presents the packet reordering probability. We see from the figure that the packet reordering probability is decreased by a factor of approximately 10 times compared to the original scheme. For instance, when the traffic load is $\lambda = 0.9$, the packet reordering probability is around 10^{-4} for a buffer size $W = 10N$ (in contrast to 10^{-3} in Fig. 3.4). When $W = 23N$, the packet reorder probability is so miniscule for $\lambda = 0.9$ that we did not observe any packet reordering in our simulations.

Fig. 3.6 presents the average delay, demonstrating a significant improvement over the original approach. It can be seen from Fig. 3.6 that the average delay is now *almost the same* to the average delay of the basic LBS, no matter how large buffer size we use. In other words, by using the hash-grouping technique, our scheme achieves an amazing tradeoff: it is expected to bound the packet reordering probability to an arbitrarily small value with a linear, i.e., $O(N)$, buffer size and a negligible average re-sequencing delay. We also see that the average delay of the scheme with hash-grouping on the log-diagonal and diagonal traffic matrices is much lower compared to the basic RS-LBS scheme. Overall, it is clear that hash-grouping significantly improves our original scheme with a minor incremental memory cost.

3.6 Conclusion

In this chapter, we showed that the amount of packet reordering that can occur in the LBS is actually quite limited. This means that packet ordering can be ensured simply by employing reordering buffers at the switch outputs. In particular, we formally bound the worst-case amount of time that a packet has to wait in these output reordering buffers before it is guar-

anteed to be ready for in-order departure with high probability, and we prove that this bound is linear with respect to the switch size. This linear bound is significant because previous approaches can add quadratic or cubic delays to the load-balanced switch. Further, we presented hash-grouping strategies at the switch outputs that can further reduce the average packet waiting times at the output reordering buffers. We showed experimentally that our packet reordering approach significantly outperforms existing load-balanced switch architectures. Finally, we believe that our analytical framework and main theoretical results are applicable to other load-balancing applications such as routing over symmetric data center fabrics.

CHAPTER 4

SRS: SAFE RANDOMIZED LOAD-BALANCED SWITCHING BY DIFFUSING EXTRA LOADS

4.1 Overview

As mentioned in Chapter 1, one simple approach for ensuring correct packet order in an application flow, which is all that matters, is randomized load-balancing. This approach, called Application Flow-Based Routing (AFBR) algorithm [5], is based on the following insight: To prevent harmful effects in TCP performance due to out-of-order packets, only packets belonging to the same application flow (e.g. a TCP/IP flow) have to depart from their output port in order. This can be achieved by forcing all packets that belong to the same application flow to go through the same intermediate port. In doing so, all packets belonging to the same application flow are guaranteed to take the same path through the switch, which avoids reordering among them. The selection of intermediate port can be easily achieved by hashing on the header field of every packet (source and destination IP addresses, source and destination ports, and protocol identification) to obtain a value from 1 to N. Hence this approach is nicknamed *TCP hashing*. Although simple and intuitive, the main drawback of TCP hashing is that stability cannot be guaranteed.

In this chapter, we investigate the sources of instability in the TCP hashing approach in order to derive mechanisms that can mitigate them. In particular, at the first switching stage, each input port i is only connected to an intermediate port once every N time slots, or equivalently at $\frac{1}{N}$ of the line rate, via a deterministic connection pattern. Persistent overloading occurs at an input port when the arrival rate of packets hashed to the same intermediate port exceeds $\frac{1}{N}$ of the line rate for a long period of time, which can occur depending on the mix of flow sizes and durations in the group of flows that gets randomly

hashed to route through the same intermediate port. For example, such persistent overloading can happen if there is a long-lived elephant flow in their midst. Although the notion of instability used here is a practical one (with respect to the limited packet buffer a switch has on each input or intermediate port), we will explain that TCP hashing could become unstable also under a theoretical and more restrictive notion of stability called rate-stability [45]. In comparison, the approach that we propose in this paper is provably rate-stable.

Similarly, at the second switching stage, each intermediate port m is also only connected to an output port j at $\frac{1}{N}$ of the line rate. Packets queued at an intermediate port may come from different input ports, possibly from all N of them. Overloading occurs at an intermediate port when the arrival rate of packets destined to the same output port, from all N inputs, exceeds $\frac{1}{N}$ of the line rate.

4.1.1 Our Approach

Two Safety Mechanisms

To remedy these problems, we extend the basic flow randomization scheme with two *safety* mechanisms. First, let λ_{ij} be the arrival rate for VOQ_{ij}, the Virtual Output Queue (VOQ) of packets arriving at input port i with output destination j. Depending on the hash values of their flow identifiers, the set of TCP/UDP flows within VOQ_{ij} can be partitioned into N subsets called bins. Each bin m, $m = 1, 2, \ldots, N$, corresponds to the set of flows that are hashed (and hence need to be switched) to intermediate port m. Using a simple credit scheme that we will describe in Section 4.2.1, without any knowledge or measurement of the value of λ_{ij}, we can limit the rate at which packets in each bin are served (switched) to at most $\frac{\lambda_{ij}}{N}$. We will show in Section 4.2.1 that, the first safety mechanism, when enforced on every VOQ, ensures no overloading at any input or intermediate port, *by the "normal" (i.e., rate-limited) traffic*, under any *admissible* arrival traffic, and that it does so in the least restrictive manner in the following sense: $\frac{\lambda_{ij}}{N}$ is indeed the maximum traffic rate that can be granted to each bin safely (i.e., without overloading any input or intermediate port).

However, by limiting the service rate of each such bin at an input port to $\frac{\lambda_{ij}}{N}$, those bins with traffic arrival rates exceeding that limit (e.g., bins that contain elephant flows as mentioned above) can grow in size. To ensure that these bins do not grow infinitely, thus leading to instability, we implement a second safety mechanism in which once a build-up of packets at a bin exceeds some threshold $W \geq N$ in size, we "evacuate" the excess load by *uniformly diffusing* the build-up of packets across all intermediate ports (i.e., a full-frame of N packets are uniformly spread one-to-one to the N intermediate ports). We introduce an easy-to-implement technique to ensure packet ordering when this evacuation mechanism kicks in, which involves requiring a "to-be-evacuated" bin to wait till it is safe (from packet reordering) to do so. Due to this waiting, careful scheduling is needed to coordinate an "orderly evacuation" of all bins being evacuated at any input port to ensure that every bin has a fair chance to have its backlog duly cleared, which we will explain in Section 4.2.1.

The Stability Proof

We prove that our Safe Randomization Switch (SRS) scheme can achieve 100% throughput (i.e., rate-stability), while guaranteeing packet order, under any admissible input traffic that is allowed to change rapidly dynamically over time. Proving the stability of SRS is very challenging partly because it appears hard to make the standard machinery of fluid analysis [46] work for this problem, despite our considerable efforts.

In arriving at this proof, we have invented a general methodology for proving the stability of queues. Our proof is based on the following extremal argument. Suppose SRS is not stable so that the total length $Q(t)$ of a subset of queues in SRS, as a function of time t, does not satisfy the stability condition $\lim_{t\to\infty} \frac{Q(t)}{t} = 0$. Then we define $\gamma \equiv \limsup_{t\to\infty} \frac{Q(t)}{t} > 0$. Let t_i, $i = 1, 2, \ldots$, be a sequence of times such that $\lim_{i\to\infty} t_i = \infty$ and $\lim_{i\to\infty} \frac{Q(t_i)}{t_i} = \gamma$. Starting with this time sequence t_i, by the properties of the aforementioned credit scheme (for rate-limiting "TCP hashed" traffic into each intermediate port) and the aforementioned scheduler for the orderly evacuation, and

through standard busy period arguments, we can construct another sequence of times t'_i, $i = 1, 2, \ldots$, such that $\lim_{i\to\infty} t'_i = \infty$ and $\limsup_{i\to\infty} \frac{Q(t'_i)}{t'_i} > \gamma$, which contradicts the definition of γ.

The rest of the chapter is organized as follows. In Section 4.2, we describe the SRS architecture in details. In Section 4.3, we prove the stability of the SRS architecture. In Section 4.4, we state a few important facts concerning the stability and starvation issues of SRS. In Section 4.5, we compare the average delay performance of SRS with other LBS solutions. Finally, we conclude the chapter in Section 4.6.

4.2 Design of SRS

In this work, we make the standard assumption that all incoming variable-size packets are segmented into fixed-size packets (sometimes referred to as cells), which are then reassembled when leaving the switch. Hence we consider the switching of only fixed-size packets in the sequel, and each such fixed-size packet takes exactly one time slot to transmit. We also make the standard homogeneity assumption that every input, intermediate, or output port operates at the same speed: Each can process and transmit exactly one (fixed-size) packet per time slot. We refer to this service rate as 1. Every connection made in a switching fabric also has speed of 1 (i.e., one packet can be switched per time slot). Since N connections are made by a switching fabric at any time slot, up to N packets can be switched by it during each time slot.

Since the connection patterns at both switching fabrics are deterministic periodic sequences, as explained in Section 2.2, the actions of a load-balanced switch are completely determined by its policies of scheduling packets (for switching service) at both the input ports and the intermediate ports. We describe the operations, including such scheduling policies and other supporting mechanisms, at input ports and at intermediate ports in Sections 4.2.1 and 4.2.2 respectively.

4.2.1 Operations at an Input Port

Throughout this section, whenever possible, we describe only the operations at an input port i, since those at other input ports are identical. We emphasize that all these operations are fully distributed and have a total computational complexity of $O(1)$ per packet per port when properly implemented. Readers may refer to Appendix B.3 for detailed discussions on this and other complexities (e.g., space).

Two Modes of Operations: RSP and UFS

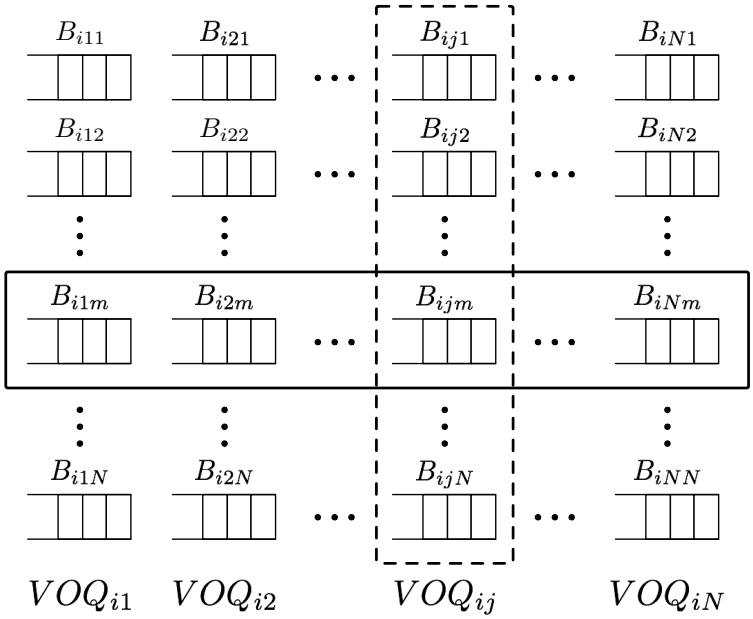


Figure 4.1: The N^2 bins at input port i.

As described earlier, for each output (destination) port $1 \leq j \leq N$, packets in VOQ_{ij} are divided into N bins via "TCP-hashing", which we denote as $B_{ij1}, B_{ij2}, \ldots, B_{ijN}$. A logical arrangement of the N^2 bins at input port i is illustrated in Figure 4.1. Each column of bins correspond to a VOQ. For example, the j^{th} column, highlighted in the figure, contains N bins B_{ij1}, B_{ij2}, ..., B_{ijN} that belong to VOQ_{ij}. Each bin is a FIFO queue: All packet arrivals to the bin are to be served in the FIFO order.

By default, for any $1 \leq m \leq N$, packets in bin B_{ijm} are routed through the intermediate

port m. We refer to this mode of operation as the Random Single Path (RSP) mode. A bin is in the *RSP mode* by default, unless it enters the other mode of operation called the *UFS mode* (described next) due to exceeding its rate limit, as mentioned earlier. With the RSP mode, all packets belonging to the same application (TCP or UDP) flow are routed along the *same path* through the switch, thereby ensuring their packet order. As explained earlier, we *limit*, the rate at which packets in each bin B_{ijm} can be served under the RSP mode to $\frac{\lambda_{ij}}{N}$, where λ_{ij} is the arrival rate of VOQ_{ij}. This rate-limiting is achieved using a simple credit-based mechanism (to be described in Section 4.2.1), in which a bin B_{ijm} is eligible for service under the RSP mode, if and only if the bin is in the RSP mode (i.e., has not entered the UFS mode) and has enough credit left to pay for the (RSP) service; we call such a bin *RSP-ready*.

When the packet arrival rate to a bin B_{ijm} exceeds this rate limit $\frac{\lambda_{ij}}{N}$, the nonconforming packets have to be queued at the bin, due to having no credit left to pay for their service under the RSP mode, and the length of the queue can become very long. Our solution is, once the queue length reaches a threshold W ($\geq N$), the switch will *eventually* allow all packets in the bin to "evacuate" through all N intermediate ports simultaneously, one frame (of N packets) at a time and hence at a very high rate, until the queue is cleared. We refer to this mode of operation as UFS, named after the prior work (described in Section 2.3.2) that serves packets within each VOQ frame by frame to avoid packet reorder [5]. As discussed earlier, when the service of a bin is switched from the RSP mode to the UFS mode, care needs to be taken so that packet reorder will not happen during the transition, which we will elaborate on in Section 4.2.1. In the meantime, we simply call a bin *UFS-ready* when it is safe (from packet reorder) to do so. Note that SRS does not inherit the long buffering delay issue from UFS for low-rate VOQs, because as mentioned earlier, it uses UFS only for "evacuating" a bin (consisting of a subset of flows in a VOQ) that has too many (rather than too few) packets in it.

Algorithm 1 The "master" bin scheduling policy at input port i.

When connected to intermediate port 1**:**

1: **if** at least one bin is UFS-ready **then**
2: Transmit the HOL frame (of N packets) of a UFS-ready bin (say B_{ijm}) in the next N time slots;
3: **else**
4: Switch packets in the RSP mode in the following N time slots, as described in Algorithm 2;

The "master" bin scheduling policy, which governs the order in which these N^2 bins are serviced (by the first switching fabric), is shown in Algorithm 1. It essentially states that, when there are both RSP-ready and UFS-ready bins waiting for their respective services, UFS-ready bins take priority. This policy makes sense because UFS kicks in only when a bin has a very long queue and needs to be "evacuated" quickly. However, as we will elaborate in Section 4.4, it may unfairly starve (i.e., deny service to) certain bins while sparing others, when the switch is persistently overloaded, although ideally it should starve every bin in a proportional fair way.

The "pseudocode" of this policy is shown in Algorithm 1. Whenever the input port i is connected to intermediate port 1 (by the first switching fabric), it checks whether one or more of these N^2 bins are UFS-ready. If so, the input port i selects one of the UFS-ready bins – according to the aforementioned "orderly evacuation" policy that we will elaborate on in Section 4.2.1 – for a full frame of UFS service: It transmits the HOL frame (i.e., the first N packets) of the selected bin in the next N time slots to the intermediate ports $1, 2, \ldots, N$ respectively.

Otherwise, input port i instead serves packets in the RSP mode during the next N time slots as follows. Recall that the input port i is connected to intermediate port $1, 2, ..., N$ respectively in next N time slots. This corresponds to rows $1, 2, \ldots, N$ (of bins) in Figure 4.1 taking turns to receive a unit (packet) of switching service. When it is the turn of row m (highlighted in Figure 4.1 and containing bins $B_{i1m}, B_{i2m}, \ldots, B_{inm}$), to receive

service, if one or more of these bins are both RSP-ready and non-empty, one such bin will be selected – in a round-robin manner – to receive RSP service (i.e., to have its HOL packet switched to intermediate port m) during this time slot. This round-robin scheduling can be implemented by maintaining, for each "row" m, a linked list of non-empty RSP-ready bins, and the computational complexity of this implementation is only $O(1)$ per packet per port, as we will elaborate in Appendix B.3. Note that, adopting the round-robin policy here for scheduling non-empty RSP-ready bins (in each row m) is for optimizing the overall delay performance of the switch, and for providing a certain degree of fairness to these bins. It is not essential for ensuring switch stability: Our rate-stability proof assumes only that this scheduling policy is work-conserving in the sense the input port i must serve a non-empty RSP-ready bin in row m if at least one such bin exists.

UFS Waiting and Evacuation Phases for Packet Reorder Avoidance

Recall that when the queue length of a bin B_{ijm} reaches or exceeds the aforementioned evacuation threshold W, its service mode is changed to UFS (from RSP). When this happens, B_{ijm} is no longer eligible for RSP service, even if it still has unused credits, until its queue is eventually cleared by UFS. In this case, the intermediate port m is informed of this change. This notification can be piggybacked to the next packet (RSP or UFS) destined for intermediate port m, and hence does not have to consume a time slot.

This bin B_{ijm} is however not eligible for the UFS service right away (i.e. not UFS-ready) for the following reason. One or more packets sent earlier from the same input bin B_{ijm} to the intermediate port may still be queued at intermediate port m, and more specifically at intermediate bin H_{ijm}, as will be explained in Section 4.2.2. Suppose B_{ijm} is allowed to start receiving UFS service right away and sends out one or more UFS frames to the intermediate ports. Because UFS packets also take strict priority over RSP packets at intermediate ports, as we will explain in Section 4.2.2, those UFS packets sent to intermediate port m from B_{ijm} may arrive at and then depart from the output port j before those

RSP packets queued in H_{ijm} do, causing packet reorder. We refer to the status of bin B_{ijm} at this moment as in the *UFS waiting phase*, in the sense that it has entered the UFS mode, but is not yet eligible for UFS service.

When the aforementioned intermediate bin H_{ijm} has been cleared at intermediate port m, a notification message is sent back to input port i. Upon receiving the notification, B_{ijm} exits the UFS waiting phase, becomes UFS-ready, and joins the ranks of other UFS-ready bins for the "orderly evacuation". We say that the bin B_{ijm} enters the *UFS evacuation phase* at this moment. By waiting for H_{ijm} to clear before evacuating B_{ijm}, which prevents the reordering between an earlier RSP packet and a later UFS packet within the same VOQ, SRS ensures correct packet order in every VOQ because, as discussed earlier, reordering cannot happen between two RSP packets or two UFS packets within the same VOQ. Note the overhead caused by such notifications is quite small, considering that even an excessively overloaded bin (say with a traffic rate close to 90% of the VOQ it belongs to) triggers such a notification no more frequent than once every $O(W)$ time slots, where $W \geq N$ is the aforementioned UFS evacuation threshold.

So far, our description of the operations at an input port i is complete except for the following two critical components: (i) the credit-based mechanism for limiting the rate at which each bin B_{ijm} can receive (switching) service under the RSP mode to $\frac{\lambda_{ij}}{N}$, where λ_{ij} is the traffic arrival rate to VOQ_{ij}, and (ii) the scheduling policy for ensuring the "orderly evacuation" of UFS-ready bins. They will be described in the next two sections respectively.

Credit-Based Mechanism for RSP Rate-Limiting

In this section, we describe the aforementioned credit-based mechanism for limiting the rate, at which each bin B_{ijm} can receive switching service under the RSP mode, to $\frac{\lambda_{ij}}{N}$. Before we do so, however, we need to first explain the rationale behind setting this rate limit to $\frac{\lambda_{ij}}{N}$. Recall that RSP is the preferred and default mode of operation due to its low

buffering delay, so we would like to make this rate limit as high as possible. Our rationale for this rate-limit is simple yet subtle: *For any* $1 \leq i, j, m \leq N$, $\frac{\lambda_{ij}}{N}$ *is the highest RSP traffic rate the input port* i *can grant, under any "nondiscriminatory policy" (i.e., with a policy statement that does not "discriminate against" any particular values of* i, j, m*), to the bin* B_{ijm} *without risking compromising the rate-stability of the set of bins that buffer packets destined for the output port* j*, at the intermediate port* m*.* We will elaborate on the details and the subtleties of this rationale in Appendix B.1.

It is actually more appropriate to consider this objective of rate-limiting a fair resource allocation scheme: Under this scheme, for any $1 \leq i, j \leq N$, the switch provides almost equal amount of RSP service to the N bins B_{ij1}, B_{ij2}, ..., B_{ijN} that traffic in VOQ_{ij} splits into (via TCP-hashing). How to perform such a fair resource allocation, using various token bucket, leaky bucket, credit counter primitives, and their combinations, has been thoroughly studied for more than three decades [47, 48, 49, 50, 51]. In fact, many techniques for accomplishing this fair resource allocation task, all of which are slight variants of one another and provide similar or identical guarantees, can be pieced together using these "off-the-shelf" primitives.

Algorithm 2 Scheduling RSP packets at input port i

When connected to intermediate port $\mathbf{m = 1, 2, \ldots, N}$**:**

1: Pick an nonempty RSP-ready bin B_{ijm} with $C^R_{ijm} \geq 1$, in round-robin order, from $\{B_{i1m}, B_{i2m}, \ldots, B_{iNm}\}$ (i.e., bins in row m as highlighted in Figure 4.1);
2: **if** such a B_{ijm} exists **then**
3: Switch the HOL packet of B_{ijm};
4: Update RSP credit counters per Algorithm 3;
5: **else**
6: Idle;

We piece together one that is simple to state, cheap to implement, and low in computational complexity ($O(1)$ per packet per port), but do not consider it a contribution of this work. It is a credit redistribution mechanism, shown in Algorithm 2 and Algorithm 3. In this mechanism, each bin B_{ijm} has a credit counter C^R_{ijm} associated with it. As shown in

Algorithm 3 Updating RSP credit counters at input port i

1: **Initialize:** Set all RSP credit counters C^R_{ijm}, $j, m = 1, 2, \ldots, N$ to a positive integer constant $C \geq 1$;

After switching the HOL packet of $\mathbf{B_{ijm}}$, for $\mathbf{i, j, m = 1, 2, \ldots, N}$:

2: Decrement counter C^R_{ijm} by 1;
3: Increment counters $C^R_{ij1}, C^R_{ij2}, \ldots, C^R_{ijN}$ each by $\frac{1}{N}$;

Line 1 of Algorithm 3, initially all credit counters are initialized to a positive constant C. Once an RSP-ready bin (say B_{ijm}) is chosen for service in the RSP mode – in the round-robin fashion as described above (Line 1 in Algorithm 2) – 1 unit of credit is subtracted from C^R_{ijm} (Line 2 in Algorithm 3), and $\frac{1}{N}$ unit of credit is deposited to each of the N credit counters associated respectively with the N bins in the j^{th} column highlighted in Figure 4.1 (Line 3 in Algorithm 3). In other words, the unit of credit paid by B_{ijm} for the RSP service, will be evenly distributed to all N bins belonging to VOQ_{ij}, including B_{ijm} itself. We will prove in Appendix B.5 that this credit redistribution mechanism provides the following fair resource allocation guarantee.

Lemma 1. *Let $D^R_{ijm}(t)$, $i, j, m = 1, \ldots, N$, be the cumulative number of packet departures from B_{ijm} in RSP mode by time slot t. Then for any two different input port bins in the same VOQ, say B_{ijm} and $B_{ijm'}$ in $VOQ(i, j)$ and for any time $t > 0$, we have*

$$|D^R_{ijm}(t) - D^R_{ijm'}(t)| \leq NC \tag{4.1}$$

It follows as an immediate "corollary" from this lemma that during any time interval $[t_1, t_2]$ that is "long enough," the respective average rates at which RSP service is provided to two different bins belonging to the same VOQ is roughly the same. This is precisely the aforementioned fair resource allocation objective we would like to achieve. While this credit mechanism is easy to describe, it is very computationally expensive ($O(N)$ per packet per port) to implement. In Appendix B.3, we will describe a low complexity ($O(1)$ per packet per port) algorithm that provides almost the same guarantee.

Orderly Evacuation in UFS Mode

Recall there are altogether N^2 bins at input port i, and many of them can be UFS-ready (i.e., in the UFS evacuation phase) at the same time, especially when the traffic load is heavy. As discussed in Section 4.1.1, these UFS-ready bins need to be evacuated (via UFS) in an orderly fashion. The idealized objective of this orderly evacuation is that, for any bin, once it enters the UFS evacuation phase, its queue length should be strictly non-increasing over time despite new packet arrivals to this bin, until it exits the UFS mode (after its queue length drops under N). This objective is however unrealistic in practice due to the following service granularity restriction: The smallest unit of UFS service is a frame (of N consecutive time slots), and while a bin is being served during these N time slots, queue lengths of some other UFS-ready bins can increase due to new packet arrivals. Hence our realistic objective is to ensure that the queue length of any UFS-ready bin is roughly non-increasing over time except for a fluctuation caused by this service granularity restriction.

This objective again can be formulated as a proportional fair resource (traffic rate) allocation problem: to serve a bin at a rate no smaller than its average packet arrival rate. Again, we achieve this objective using a scheduler comprised of aforementioned "off-the-shelf" primitives, but do not consider this scheduler a contribution of this work. More specifically, this scheduler achieves the following guarantee (that the bin length is roughly non-increasing), which we prove in Appendix B.8.

Lemma 2. *Let $B_{ijm}(t)$, $i, j, m = 1, \ldots, N$, denote the queue length of the bin B_{ijm} at time slot t. If B_{ijm} remains in UFS evacuation phase during a time interval $[t_1, t_2]$, we must have*

$$B_{ijm}(t_2) - B_{ijm}(t_1) \leq N^3$$

The basic idea of this scheduler is to keep track of *increases in backlog* to a bin *after* it has become UFS-ready. This is achieved by associating a UFS "pressure" counter with

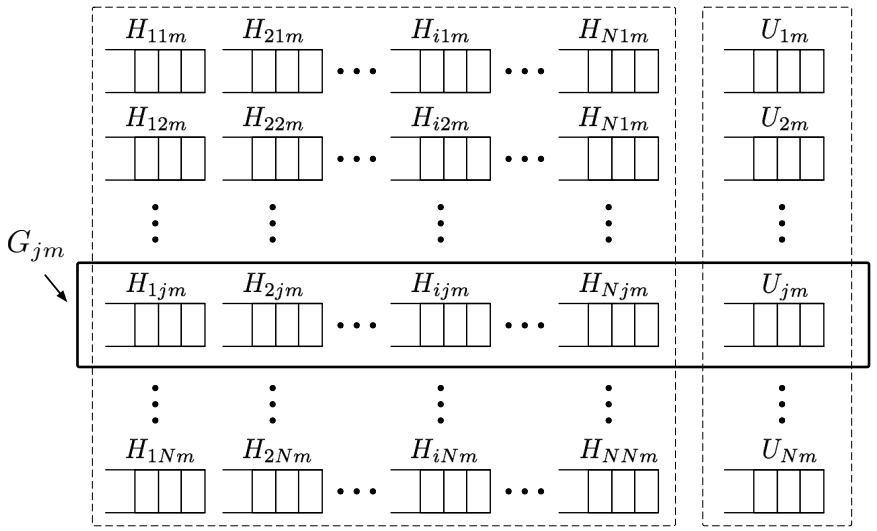


Figure 4.2: How bins are served at intermediate port m. Queue group G_{jm}, highlighted here, will be defined in Section 4.3.

each bin that keeps track of the number of new packet arrivals since the bin has become UFS-ready (i.e., the "pressure build-up"), minus the "pressure relief" in the form of UFS service provided to this bin. Whenever the scheduler needs to pick a UFS frame to serve next, it will pick the HOL frame of the bin with the highest pressure counter value. The detailed design of this scheduler is shown in Appendix B.2. The computational complexity of this scheduler is $O(1)$ per packet per port, as we explain in Appendix B.3.

4.2.2 Operations at an intermediate port

In SRS, intermediate ports play a passive role in packet and frame scheduling. They mostly follow the RSP and UFS scheduling decisions made by the input ports. More specifically, like input ports, intermediate ports also prioritize UFS frames over RSP packets. In addition, intermediate ports serve UFS frames in the order dictated by the input ports, which, as far as the intermediate ports are concerned, is the FIFO order.

We describe operations at an intermediate port m: Operations at other intermediate ports are identical. As shown in Figure 4.2, at intermediate port m, N^2 RSP bins H_{ijm} are maintained for $i, j = 1, 2, \ldots, N$. Bin H_{ijm} buffers RSP packets switched from input

port i that are destined for output port j. It is clear that all these packets come from input bin B_{ijm}. In addition, N UFS bins U_{jm}, for $j = 1, 2, \ldots, N$, are maintained for buffering UFS packets sent to it from all N input ports, with all UFS packets destined for output port j appended to the end of the bin U_{jm}. Whenever intermediate port m is connected to output port j, the intermediate port m first checks if the corresponding UFS queue U_{jm} is non-empty. If so, it *first* serves its HOL packet. Otherwise, it serves a RSP packet from one of $H_{1jm}, H_{2jm}, \ldots, H_{Njm}$ in the round-robin[1] order. Finally, when a bin H_{ijm} is cleared and B_{ijm} is in the UFS waiting phase, a notification to input port i is triggered.

4.2.3 Variation based on input port load

In this section, we describe a variant of the baseline SRS architecture that can improve its delay performance when the traffic load is high, as will be shown in Section 4.5. Its basic idea is that if the total traffic arrival rate to an input port is very high, the switch simply serves it using the UFS scheme. Note this UFS scheme, which serves each VOQ frame by frame, is different than the UFS mode, which serves each bin frame by frame. We call this variant "SRS-UFS". A high-level overview of this variant is as follows:

(i) When a packet arrives, besides being hashed into a bin B_{ijm} as in SRS, it is also inserted into a "shadow" VOQ Z_{ij}. When a packet is removed from B_{ijm}, it is also removed from its shadow VOQ Z_{ij} (and vice versa when Z_{ij} is served by UFS as described next).

(ii) Each input port tracks its traffic arrival rate, using a lightweight measurement mechanism such as "sample and count". When the arrival rate to an input port i exceeds some threshold $\bar{\rho}$, say 0.75, the input port i stops "TCP-hashing" its traffic into the N^2 B_{ijm} bins, and instead demultiplexes all future packet arrivals to their respective shadow VOQs (there are N of them). These shadow VOQs will soon be served via

[1]Note that our stability proof, shown in the next section, assumes only that this service discipline is work-conserving. We choose round-robin scheduler simply because it is computationally cheap ($O(1)$ per packet per port) to implement, as we explain in Appendix B.3.

UFS. However, before this "SRS to UFS" transition happens, the input port i has to wait until all N^2 corresponding H_{ijm} queues at the N intermediate ports are cleared, to prevent packet reordering.

(iii) Once an input port starts to serve all its VOQs via UFS, it continues to do so until the traffic arrival rate to the input port drops below a smaller threshold $\underline{\rho}$ (than $\bar{\rho}$), say 0.65, at which point the switch transitions back to serving the traffic via the baseline SRS. Keeping a "safe distance" between these two thresholds prevents the switch from transitioning back and forth frequently between the baseline SRS and the UFS scheme, when the traffic arrival rate to an input port fluctuates around the higher threshold $\bar{\rho}$. It also improves the tolerance of the switch to the inaccuracies in tracking the traffic arrival rates to the input ports using a lightweight measurement mechanism.

4.3 Stability Analysis

In this section, we prove that SRS can achieve 100% throughput[2]. Equivalently, we prove that the total backlog in the B_{ijm}, H_{ijm} or U_{jm} bins does not accumulate at a positive rate over time, even when the switch is 100% loaded. This notion of (queue) stability is known as rate-stability [45], because it implies that the long-run average of packet departure rate is equal to that of packet arrival rate when the switch is no more than 100% loaded.

At the first glance, this stability is guaranteed since as queues grow longer the algorithm switches to UFS, which is known to be (rate) stable. However, the short stability explanation/proof in the UFS paper [5] does not apply to SRS for several reasons. Chief among them is that, in SRS, an input port bin can start transmitting in the UFS mode only after its corresponding intermediate port bin is cleared, which may never happen with poorly designed safety mechanisms, whereas in UFS, this transmission does not have to wait. To implicitly (i.e., woven into the fabric of the stability proof) prove that this clearance

[2]The rate-stability of SRS-UFS can be derived from that of SRS and of UFS.

will happen because of the properly designed safe mechanisms (fair RSP rate-limiting and orderly UFS evacuations) contributes to the length and the perceived complexities of the proof.

Recall that $B_{ijm}(t)$ denotes the queue length of bin B_{ijm} at time t; $H_{ijm}(t)$ and $U_{jm}(t)$ are defined similarly. Our main result is:

Theorem 2.

$$(a) \quad \lim_{t\to\infty} \frac{B_{ijm}(t)}{t} = 0, \quad i,j,m = 1,\ldots,N$$
$$(b) \quad \lim_{t\to\infty} \frac{H_{ijm}(t)}{t} = 0, \quad i,j,m = 1,\ldots,N$$
$$(c) \quad \lim_{t\to\infty} \frac{U_{jm}(t)}{t} = 0, \quad j,m = 1,\ldots,N$$

for any deterministic packet arrival process that satisfies a mild admissible condition (to be stated next) and for any arbitrary initial queue lengths of B, H, and U (at time 0).

Note that in the proof, we actually assume all queues in the switch are empty at time 0. It is however not hard to extend this proof to accommodate arbitrary initial queue lengths, as will be explained at the end of Section 4.3.2.

We now state the only (mild) admissible condition that we have to exogenously impose on the traffic arrival process. Let $A_{ijm}(t)$ be the cumulative number of packet arrivals into bin B_{ijm} by time slot t (since time 0). Define $A_j(t) \equiv \sum_{i,m=1}^{N} A_{ijm}(t)$. $A_j(t)$ is the total number of packets that have arrived at all input ports by time slot t and are destined for output port j. The admissible condition, which we exogenously impose, is that, for each output port j, that there exist a constant $\lambda_j \in [0,1]$ such that

$$\lim_{t\to\infty} \frac{A_j(t)}{t} = \lambda_j \quad j = 1,2,\ldots,N \tag{4.2}$$

In other words, the long-run-average total rate of all traffic destined for output port j must exist and is no more than 1 (i.e., 100% loaded). This admissible condition is weaker than the

usual notion of admissibility, in which the long-run-average of each λ_{ij}, for $i = 1, 2, \ldots, N$, must exist.

In the worst case, up to N packets destined for output j can arrive, one at each input port, during a single time slot. Therefore, for any $t \geq 0$, we always have

$$A_j(t) \leq Nt \tag{4.3}$$

Note that, in our proof, we do assume another admissible condition that at most one packet can arrive at each input port in a single time slot. This condition is however imposed "endogenously" by the maximum rate of the network link and the clock speed of the input line card circuitry, not "exogenously" by us.

In the following, we first develop a property of the packet arrival process $A_j(t)$ in the form of a general lemma, and then describe some important queuing dynamics of the switch that result from the aforementioned RSP credit redistribution mechanism and the aforementioned UFS orderly evacuation scheduler. Then we prove Theorem 2 in Section 4.3.2.

4.3.1 System dynamics and notations

We develop only the dynamics of queues that hold packets destined for (i.e., associated with) an arbitrary (but fixed) output port j; Queues associated with any other output port have the same dynamics. To facilitate the following presentation, for $i, m = 1, 2, \ldots, N$, we define a set of queue groups as follows

$$B_j \equiv \left\{ B_{ijm} \,\middle|\, i, m = 1, 2, \ldots, N \right\}$$

$$B_{ij} \equiv \left\{ B_{ijm} \,\middle|\, m = 1, 2, \ldots, N \right\}$$

$$G_{jm} \equiv \{U_{jm}\} \cup \left\{ H_{ijm} \,\middle|\, i = 1, 2, \ldots, N \right\}$$

Let $B_j(t) \triangleq \sum_{i,m=1}^{N} B_{ijm}(t)$ be the total number of packets in queue group B_j at time slot t. Similarly we define $G_{jm}(t) \triangleq \sum_{i=1}^{N} H_{ijm}(t) + U_{jm}(t)$, $m = 1, \ldots, N$.

As mentioned earlier, throughout this paper, time is slotted and is numbered by nonnegative integers, and we use the terms "time" and "time slot" interchangeably. By convention, time starts at (slot) 0 and packets start to arrive at or after (slot) 1. When we say "at/by time (slot) t", we mean "at/by *the end of* time slot t". For example, the queue length of B_{ijm} at time t refers to that at the end of time slot t, which accounts for any (packet) arrival and/or any departure that has happened during time slot t.

Arrival process dynamics

In this section, we state a purely mathematical lemma (i.e., has nothing to do with switching or networking by itself) that applies to any deterministic arrival process whose long-run average converges to a constant, including the aforementioned $A_j(t)$.

Lemma 3. *Let $\{X(t),\ t \geq 0\}$, be an arbitrary deterministic time series. If there exists a constant $\lambda \in \mathbb{R}$ such that $\lim_{t\to\infty} \frac{X(t)}{t} = \lambda$, then for any $\epsilon > 0$ and $p \in (0, 1]$, there exists a constant $T_X > 0$, such that*

$$\left\{ \begin{array}{l} T \geq T_X \\ 0 \leq t_1 < t_2 \leq T \\ t_2 - t_1 \geq pT \end{array} \right\} \Rightarrow\ X(t_2) - X(t_1) \leq (\lambda + \epsilon)(t_2 - t_1) \tag{4.4}$$

Proof. As $\lim_{t\to\infty} \frac{X(t)}{t} = \lambda$, for $\epsilon' = p\epsilon/4$, there exists a constant $T' > 0$, such that

$$t \geq T' \Rightarrow \left| \frac{X(t)}{t} - \lambda \right| < \frac{\epsilon'}{2}$$

Let $T_X = \max\left(\frac{2T'}{p}, \frac{T'}{p} + \frac{X(T')}{\frac{\epsilon}{2}\cdot p}\right)$.

As $T \geq T_X \geq \frac{2T'}{p}$ and $t_2 \geq pT$, we have $t_2 - T' \geq t_2 - pT/2 \geq pT/2$. Then we must have $T' \leq (1 - \frac{1}{2}p)t_2$ or equivalently $T' \leq \frac{2-p}{p}(t_2 - T')$. Otherwise, we'll have

$t_2 \geq T' + \frac{1}{2}pT > (1 - \frac{1}{2}p)t_2 + \frac{1}{2}pT$, which implies $t_2 > T$. But this contradicts our assumption that $t_2 \leq T$. Note that $t_2 \geq pT > T'$, we must have,

$$\begin{aligned}
X(t_2) - X(T') &< (\lambda + \frac{\epsilon'}{2})t_2 - (\lambda - \frac{\epsilon'}{2})T' \\
&= (\lambda + \frac{\epsilon'}{2})(t_2 - T') + \epsilon' T' \\
&\leq (\lambda + \frac{\epsilon'}{2})(t_2 - T') + \epsilon' \frac{2-p}{p}(t_2 - T') \\
&= \left(\lambda + \frac{\epsilon'}{2} + \frac{2-p}{p}\epsilon'\right)(t_2 - T') \\
&= \left(\lambda + \frac{4-p}{2p}\epsilon'\right)(t_2 - T') \\
&\leq \left(\lambda + \frac{2}{p}\epsilon'\right)(t_2 - T') \\
&= \left(\lambda + \frac{\epsilon}{2}\right)(t_2 - T')
\end{aligned}$$

Furthermore, we have $t_2 \geq pT \geq pT_X \geq \frac{X(T')}{\epsilon/2} + T'$, which implies $X(T') - X(t_1) \leq X(T') \leq \frac{\epsilon}{2}(t_2 - T')$. Hence,

$$\begin{aligned}
X(t_2) - X(t_1) &= (X(t_2) - X(T')) + (X(T') - X(t_1)) \\
&= \left(\lambda + \frac{\epsilon}{2}\right)(t_2 - T') + \frac{\epsilon}{2}(t_2 - T') \\
&\leq (\lambda + \epsilon)(t_2 - T') \\
&\leq (\lambda + \epsilon)(t_2 - t_1)
\end{aligned}$$

□

Remark: We will show in Appendix B.4 that an arrival process could be extremely bursty yet still has a long-run average rate. Hence, for such an arrival process, a time interval $[t_1, t_2]$ has to be "very long" (at least a constant fraction of T here) for its average rate during the interval to be bounded by $\lambda + \epsilon$.

RSP rate-limiting dynamics

How RSP rate-limiting mechanism affects the queuing dynamics of the switch is captured in the following two lemmas.

Lemma 4. *For any two intermediate ports m and m', we have $|I_{jm}(t) - I_{jm'}(t)| \leq N^2C + 1$.*

Here $I_{jm}(t)$ is defined as the cumulative number, by time slot t, of packets that arrive at the queue group G_{jm}, the j^{th} row of bins at intermediate port m as shown in Figure 4.2. In other words, $I_{jm}(t)$ accounts for all packets destined for output port j that arrive at intermediate port m during time interval $[0, t]$. Lemma 4 states that, the set of packets destined for output j that depart from all input ports of the switch during $[0, t]$ (denoted as $\Phi_j(t)$), arrive at every intermediate port in roughly equal numbers. Its proof, based on Lemma 1, is provided in Appendix B.6.

We denote as $D_j(t)$ the size of this set $\Phi_j(t)$, i.e., $D_j(t) \triangleq |\Phi_j(t)|$. The following lemma states that if a "long enough" busy period of queue group G_{jm} starts at t_1 and contains $t_2 > t_1$, then $D_j(t_2) - D_j(t_1)$, the number of packets that belong to the set $\Phi_j(t)$ and have departed from the input ports during $[t_1, t_2]$, is roughly equal to $t_2 - t_1$. Intuitively, this is because (i) Lemma 4 implies these $D_j(t_2) - D_j(t_1)$ packets arrive at at every intermediate port also in roughly equal numbers, and (ii) at least $\frac{1}{N}(t_2 - t_1)$ such packets must arrive at G_{jm} during $[t_1, t_2]$ to keep it continuously busy. Its formal proof is provided in Appendix B.7.

Lemma 5. *For $0 \leq t_1 < t_2$, if there exists an intermediate port m such that $G_{jm}(t_1) = 0$ and $G_{jm}(t) > 0$ for any $t \in [t_1 + 1,\ t_2]$, we have $D_j(t_2) - D_j(t_1) \geq (t_2 - t_1) + NG_{jm}(t_2) - 5N^3C$.*

UFS orderly evacuation dynamics

The effect of the aforementioned "orderly evacuation" scheduler on the dynamics of input port bins is characterized in Lemma 2 (see Section 4.2.1). For proving Theorem 2 however,

we need the following lemma that is a slight generalization of Lemma 2.

Corollary 1. *If B_{ijm} has never been in UFS waiting phase during $[t_1, t_2]$, we have*

$$B_{ijm}(t_2) - B_{ijm}(t_1) \leq W^3$$

Proof. If B_{ijm} is in RSP mode at time t_2, we have $B_{ijm}(t_2) < W$ and thus

$$B_{ijm}(t_2) - B_{ijm}(t_1) \leq B_{ijm}(t_2) < W \leq W^3$$

Otherwise, if B_{ijm} is in UFS evacuation phase at t_2, it must be in UFS evacuation phase throughout $[t_1, t_2]$. By Lemma 2, we have $B_{ijm}(t_2) - B_{ijm}(t_1) \leq N^3 \leq W^3$. □

4.3.2 Proof of Theorem 2

In this section we present the proof of Theorem 2 in details. We first prove Theorem 2(a) (i.e., the stability of B bins) using the aforementioned extremal argument. Theorem 2(b) and (c) (i.e., the stability of H and U bins) are then much easier to prove, and their proofs will be presented in Appendix B.10. The following theorem implies that $\lim_{t\to\infty} \frac{B_{ijm}(t)}{t} = 0$, $i, j, m = 1, 2, \ldots, N$, which is Theorem 2(a).

Theorem 3.

$$\limsup_{t\to\infty} \frac{B_j(t)}{t} = 0 \quad j = 1, 2, \ldots, N$$

To prove Theorem 3, we need the following technical lemma concerning the busy period of a bin.

Lemma 6. *Suppose B_{ijm} remains in UFS waiting phase throughout time slots t_1 to t_2, there must exist a time slot $t_0 \leq t_1$ such that $G_{jm}(t_0) = 0$ and $G_{jm}(t) > 0$ for any $t \in [t_0 + 1, t_2]$.*

Proof. Since H_{ijm} should be non-empty whenever B_{ijm} is in the UFS waiting phase, we have $G_{jm}(t) \geq H_{ijm}(t) > 0$ for any $t \in [t_1, t_2]$. Note that initially we have $G_{jm}(0) = 0$.

Thus there must exist a time slot $t_0 \leq t_1$ such that $G_{jm}(t_0) = 0$ and $G_{jm}(t) > 0$ for any $t \in [t_0 + 1,\, t_2]$. □

Intuition to the proof of Theorem 3

We provide some intuitions to the proof of Theorem 3 in this section, deferring the formal proof to Section 4.3.2. In describing these intuitions, we will liberally use vague phrases such as "roughly", "not by much", and "very large". To rid the formal proof of this vagueness accounts for the bulk of its length and perceived complexities. Following the "limsup argument" mentioned in Section 4.1.1, we assume

$$\limsup_{t\to\infty} \frac{B_j(t)}{t} = \gamma > 0 \tag{4.5}$$

This implies that given any $\epsilon > 0$ we have

(i) There exists an integer $T' \geq 0$, such that $t \geq T' \Rightarrow \frac{B_j(t)}{t} < (\gamma + \epsilon)$.

(ii) Given any integer $T \geq 0$, there exists $t \geq T$ such that $\frac{B_j(t)}{t} > (\gamma - \epsilon)$.

We fix a "tiny" ϵ (that is "much smaller" than γ), and T' based on this choice of ϵ. Let T_2 be a "very large" positive integer (with respect to T' and a few other large constants) such that

$$\frac{B_j(T_2)}{T_2} > (\gamma - \epsilon) \tag{4.6}$$

and $T_2(\gamma - \epsilon)$ is also "very large". Starting with this inequality, we would like to show that there exists T_1 that satisfies $T' < T_1 < T_2$, such that

$$B_j(T_1) > (\gamma + \epsilon)T_1 \tag{4.7}$$

a contradiction to Implication (i) of Equation (4.5). The argument we use to establish Inequality (4.7) is straightforward: It is literally "something just doesn't add up". More specifically, we prove that this T_1 is smaller than T_2 by "a significant amount" (of time),

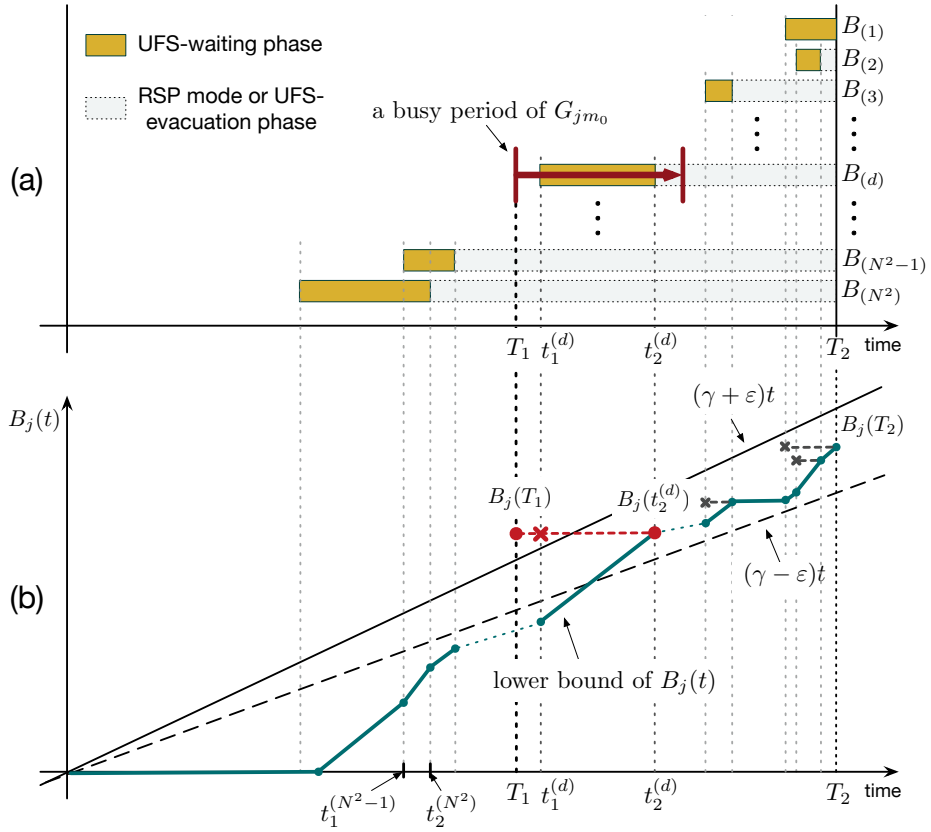


Figure 4.3: Proof of Theorem 3.

but the increase of $B_j(t)$, over this "very long" time interval $[T_1, T_2]$, is smaller than $(T_2 - T_1)\gamma - (T_2 + T_1)\epsilon$, the minimum amount needed to prevent Inequality (4.7) from happening, as follows.

Recall that the queue group B_j consists of N^2 bins, N at each input port, whose packets are destined for the output port j. They are $\{B_{ijm}\}_{i,m=1}^{N}$. Let $[t_1^{[ijm]}, t_2^{[ijm]}]$, $i, m = 1, 2, \ldots, N$, be the last UFS waiting phase of B_{ijm} before or at time T_2. If B_{ijm} has never been in the UFS waiting phase throughout $[0, T_2]$, we simply set $t_1^{[ijm]} = t_2^{[ijm]} = 0$ (the degenerated case). If B_{ijm} is still in a UFS waiting phase at T_2, we set $t_2^{[ijm]} = T_2$ (truncated by T_2). We sort these bins in the decreasing order of their $(t_2^{[ijm]})$'s, the ending times (possibly truncated or degenerated) of their last UFS evacuation phases before or at T_2, and relabel these bins as $B_{(k)}$, $k = 1, \ldots, N^2$ by the standard (reversed) order statistics notation (A more rigorous definition of this relabeling is provided in Appendix B.9). For each bin

$B_{(k)}$, we also relabel this starting and ending times as $t_1^{(k)}$ and $t_2^{(k)}$ respectively. Hence, we have $T_2 \geq t_2^{(1)} \geq t_2^{(2)} \geq \cdots \geq t_2^{(N^2)}$, and for any k, $B_{(k)}$ should never be in the UFS waiting phase throughout $[t_2^{(k)}, T_2]$. Figure 4.3 (a) shows, from top to bottom, an example instance of these N^2 bins in this sorted order. For each bin $B_{(k)}$, its last UFS waiting phase before T_2 is shown as a yellow strip; it is followed by the interval $[t_2^{(k)}, T_2]$, shown as a light grey strip (if not degenerated).

Recall that, according to Lemma 2, once a bin enters the UFS evacuation phase, its queue length will not increase "much" (i.e., by more than a constant) and may go down. Recall also that when a bin is in the RSP mode, its queue length is less than W, the threshold that would trigger the bin entering the UFS mode. Now we zoom in to see how $B_j(t)$ can grow to the very large value of (at least) $T_2(\gamma - \epsilon)$ at time T_2. For each bin $B_{(k)}$, its queue length $B_{(k)}(t)$ at time $t_1^{(k)}$, when $B_{(k)}$ just enters its last UFS accumulation phase (from the RSP mode), is at most W as just explained. In addition, $B_{(k)}(t)$ will not increase "much" (and may decrease) during $[t_2^{(k)}, T_2]$, because $B_{(k)}$ is either in the UFS evacuation phase or in the RSP mode at any time $t \in [t_2^{(k)}, T_2]$. In other words, for each $B_{(k)}$, it can grow its queue length (of no more than W at time $t_1^{(k)}$) at a rate no more than 1 (packet per time slot) – and contribute to the growth of $B_j(t)$ – "pretty much" *only* during its "yellow strip" $[t_1^{(k)}, t_2^{(k)}]$.

Based on this "growth picture" of $B_j(t)$, we establish a lower bound of $B_j(t)$ for $t \in [0, T_2]$ by "walking back in time" starting from $t = T_2$. Figure 4.3 (b) shows this lower bound curve. The rightmost point on this curve $(T_2, B_j(T_2))$ lies above the line $y = (\gamma - \epsilon)x$ due to Implication (ii) of Inequality (4.5). As shown in Figure 4.3 (b), within each interval $[t_1^{(k)}, t_2^{(k)}]$ (for $k = 1, \ldots, N^2$), this lower bound curve drops as t *decreases*; otherwise (i.e., outside all these intervals) this curve is flat. The rate (i.e., slope) of this drop follows the aforementioned fact that each queue $B_{(k)}$ grows at a rate strictly no more than 1 when t *increases*. For example, in Figure 4.3 (b), the slope of the curve is 2 during the interval $[t_1^{(N^2-1)}, t_2^{(N^2)}]$ because the bins $B_{(N^2-1)}$ and $B_{(N^2)}$ can each potentially grow

at the maximum rate of 1 during this interval.

Suppose, as shown in Figure 4.3 (b), $[t_1^{(d)}, t_2^{(d)}]$ is the *first* "very long" yellow strip, in the sense that none of the $d-1$ "earlier" yellow strips (i.e., $[t_1^{(k)}, t_2^{(k)}]$, for $1 \leq k \leq d-1$) is "very long". Suppose $B_{(d)}$ corresponds to $B_{i_0 j m_0}$. We know from Lemma 6 that $[t_1^{(d)}, t_2^{(d)}]$ is covered by a busy period, highlighted in Figure 4.3 (a), of the intermediate port queue group G_{jm_0}. The starting time of this busy period is precisely the aforementioned T_1 we are looking for. In Lemma 5, we proved that the average departure rate of queue group B_j is "roughly" 1 during this "very long" busy period. However, since this busy period is "very long", the average (total) arrival rate to all queues in the queue group B_j during this busy period is "roughly" $\lambda_j \leq 1$, according to Lemma 3. Hence, during the interval $[T_1, t_2^{(d)}]$, the value of $B_j(t)$ "roughly" does not increase. This effect is illustrated in Figure 4.3 (b) as a horizontal line from the point $\big(t_2^{(d)}, B_j(t_2^{(d)})\big)$ "back in time" to the point $\big(T_1, B_j(T_1)\big)$. However, since the lower bound curve does not drop much when t decreases from T_2 to $t_2^{(d)}$ (because none of the $d-1$ yellow strips before $[t_1^{(d)}, t_2^{(d)}]$ is "very long") and as just explained flattens out when t decreases from $t_2^{(d)}$ to T_1, the point $\big(T_1, B_j(T_1)\big)$ is "forced" to stick out above the curve $(\gamma+\epsilon)t$. In other words, we have $B_j(T_1) > (\gamma+\epsilon)T_1$, the Inequality (4.7) we are trying to prove.

Formal proof of Theorem 3

Proof. We prove the theorem by contradiction. Suppose there exists an output port j such that

$$\limsup_{t\to\infty} \frac{B_j(t)}{t} = \gamma > 0 \tag{4.8}$$

Note that we must have $\gamma \leq 1$ as $\limsup_{t\to\infty} \frac{B_j(t)}{t} \leq \lim_{t\to\infty} \frac{A_j(t)}{t}$. Let $f(x) \triangleq \frac{\gamma+5x}{2\left(1+\frac{1}{\gamma+x}\right)^{N^2}} - 3x$; this f is defined only for introducing the number sequence $\{a_k\}_{k=0}^{N^2}$ in (4.13). Since $f(x)$ is a continuous function of x in a neighborhood of 0 and $f(0) > 0$, there must exist an $\epsilon' > 0$ such that $f(x) > 0$ for $x \in (0, \epsilon')$. Let $\epsilon = \min(\frac{\gamma}{4}, \frac{\epsilon'}{2})$, by (4.8), there exists an

integer $T' > 0$, such that

$$t > T' \Rightarrow \frac{B_j(t)}{t} < (\gamma + \epsilon) \tag{4.9}$$

By Lemma 3, for $p = \min\left(\frac{f(\epsilon)}{2(\gamma+\epsilon)}, 1\right)$, there exists an integer $T_A > 0$, such that

$$\left\{ \begin{array}{l} T > T_A \\ 0 \le t_1 < t_2 \le T \\ t_2 - t_1 \ge pT \end{array} \right\} \Rightarrow A_j(t_2) - A_j(t_1) \le (1+\epsilon)(t_2 - t_1) \tag{4.10}$$

From (4.8), for the same ϵ, there exists another integer

$$T_2 > \max\left(T_A, \frac{16W^5C}{f(\epsilon)}, \frac{4W^5}{\gamma - \epsilon}, \frac{2(NT' + 8W^5C)}{\gamma - 3\epsilon}\right) \tag{4.11}$$

such that

$$\frac{B_j(T_2)}{T_2} > (\gamma - \epsilon) \tag{4.12}$$

Define two number sequences $\{a_k\}_{k=0}^{N^2}$ and $\{S_k\}_{k=0}^{N^2}$ as

$$\begin{aligned} a_k &= \begin{cases} f(\epsilon)T_2 + 8W^5C\left(\frac{1}{\left(1+\frac{1}{\gamma+\epsilon}\right)^{N^2}} - 1\right) & \textit{if } k = 0 \\ \frac{1}{\gamma+\epsilon}\left(1 + \frac{1}{\gamma+\epsilon}\right)^{k-1}(a_0 + 3\epsilon T_2 + 8W^5C) & \textit{if } k \ge 1 \end{cases} \\ S_k &= \sum_{i=0}^{k} a_i \qquad k = 0, 1, \ldots, N^2 \end{aligned} \tag{4.13}$$

It is not hard to verify the following properties of these two sequences.

- The following equation holds for $k = 1, 2, \ldots, N^2$

$$a_k = \frac{1}{\gamma + \epsilon} S_{k-1} + \frac{1}{\gamma + \epsilon}(3\epsilon T_2 + 8W^5C) \tag{4.14}$$

- Since $\epsilon < \epsilon'$ and $T_2 > \frac{16W^5C}{f(\epsilon)}$, we have $f(\epsilon) > 0$ and

$$a_0 \geq f(\epsilon)T_2 - 8W^5C \geq \frac{f(\epsilon)}{2}T_2$$

Note that $a_{k+1} > a_k$ for $k \geq 1$, thus for $k = 1, 2, \ldots, N^2$ we have

$$a_k \geq a_1 \geq \frac{1}{\gamma + \epsilon}a_0 \geq \frac{f(\epsilon)}{2(\gamma + \epsilon)}T_2 \geq pT_2$$

- When $k = N^2$, we have

$$S_{N^2} = \frac{\gamma - \epsilon}{2}T_2$$

As mentioned in Section 4.3.2, we relabel the input port bins $\{B_{ijm}\}_{i,m=1}^{N}$ as $\{B_{(k)}\}_{k=1}^{N^2}$ in the decreasing order of the end times of their last UFS waiting phases before T_2, and denote as $[t_1^{(k)}, t_2^{(k)}]$ the last UFS waiting phase of $B_{(k)}$ before T_2. Also to make the reasoning easier, we define a dummy bin $B_{(0)}$ with $B_{(0)}(t) \equiv 0$, and a corresponding dummy interval $t_1^{(0)} = t_2^{(0)} = T_2$. Thus, for any integer $k \in [1, N^2]$, we have $t_2^{(k)} \leq t_2^{(k-1)} \leq T_2$, and $B_{(k)}$ should never be in UFS waiting phase throughout $[t_2^{(k)}, T_2]$, as shown in Figure 4.3 (a). From Corollary 1, we know that

$$B_{(k)}(T_2) - B_{(k)}(t_2^{(k)}) \leq W^3 \quad k = 0, 1, \ldots, N^2 \tag{4.15}$$

Hence

$$\begin{aligned} B_j(T_2) &= \sum_{i,m=1}^{N} B_{ijm}(T_2) = \sum_{k=0}^{N^2} B_{(k)}(T_2) \\ &= \sum_{k=0}^{N^2} \Big(B_{(k)}(t_2^{(k)}) - B_{(k)}(t_1^{(k)}) \Big) + \sum_{k=0}^{N^2} B_{(k)}(t_1^{(k)}) \\ &\quad + \sum_{k=0}^{N^2} \Big(B_{(k)}(T_2) - B_{(k)}(t_2^{(k)}) \Big) \end{aligned}$$

$$\leq \sum_{k=0}^{N^2} \left(t_2^{(k)} - t_1^{(k)} \right) + \sum_{k=1}^{N^2} W + \sum_{k=1}^{N^2} W^3 \tag{4.16}$$
$$\leq \sum_{k=0}^{N^2} \left(t_2^{(k)} - t_1^{(k)} \right) + 2W^5$$

which implies that $\sum_{k=0}^{N^2} \left(t_2^{(k)} - t_1^{(k)} \right) \geq B_j(T_2) - 2W^5 \geq (\gamma - \epsilon) T_2 - 2W^5 \geq \frac{\gamma - \epsilon}{2} T_2 = S_{N^2} \equiv \sum_{k=0}^{N^2} a_k$. The last inequality holds since $T_2 \geq \frac{4W^5}{\gamma - \epsilon}$ (due to (4.11)). Therefore, there must exist an integer $k' \in [1, N^2]$ such that $t_2^{(k')} - t_1^{(k')} \geq a_{k'}$ (note that $t_2^{(0)} - t_1^{(0)} = 0 < a_0$). Let d be the smallest such integer; the corresponding interval $[t_1^{(d)}, t_2^{(d)}]$ is precisely the *first* "very long" yellow strip as defined in the last paragraph of Section 4.3.2. In other words, we have $t_2^{(k)} - t_1^{(k)} < a_k$ for $k = 0, 1, \ldots, d-1$ and $t_2^{(d)} - t_1^{(d)} \geq a_d$. Note that for any $k \geq d$, $B_{(k)}$ must never be in UFS waiting phase throughout $[t_2^{(d)}, T_2]$. By Corollary 1, we have $B_{(k)}(T_2) - B_{(k)}(t_2^{(d)}) \leq W^3$, $k = d, \ldots, N^2$. This inequality and (4.15) will be used in the first inequality below. Similar to (4.16), we can prove that

$$\begin{aligned}
B_j(T_2) &= \sum_{k=0}^{N^2} B_{(k)}(T_2) = \sum_{k=0}^{d-1} B_{(k)}(T_2) + \sum_{k=d}^{N^2} B_{(k)}(T_2) \\
&= \sum_{k=0}^{d-1} B_{(k)}(t_1^{(k)}) + \sum_{k=0}^{d-1} \left(B_{(k)}(t_2^{(k)}) - B_{(k)}(t_1^{(k)}) \right) \\
&\quad + \sum_{k=0}^{d-1} \left(B_{(k)}(T_2) - B_{(k)}(t_2^{(k)}) \right) \\
&\quad + \sum_{k=d}^{N^2} B_{(k)}(t_2^{(d)}) + \sum_{k=d}^{N^2} \left(B_{(k)}(T_2) - B_{(k)}(t_2^{(d)}) \right) \\
&\leq \sum_{k=0}^{d-1} W + \sum_{k=0}^{d-1} \left(t_2^{(k)} - t_1^{(k)} \right) + \sum_{k=0}^{d-1} W^3 \\
&\quad + \sum_{k=d}^{N^2} B_{(k)}(t_2^{(d)}) + \sum_{k=d}^{N^2} W^3 \\
&\leq \sum_{k=d}^{N^2} B_{(k)}(t_2^{(d)}) + \sum_{k=0}^{d-1} \left(t_2^{(k)} - t_1^{(k)} \right) + 3W^5
\end{aligned}$$

$$\leq \sum_{k=0}^{N^2} B_{(k)}(t_2^{(d)}) + \sum_{k=0}^{d-1} a_k + 3W^5C$$
$$= B_j(t_2^{(d)}) + S_{d-1} + 3W^5C \tag{4.17}$$

By Lemma 6, there exists a time slot $T_1 \leq t_1^{(d)}$ and an intermediate port queue group G_{jm} such that $G_{jm}(T_1) = 0$ and $G_{jm}(t) > 0$ for any $t \in [T_1 + 1, t_2^{(d)}]$. Since $T_2 \geq T_A$ and $t_2^{(d)} - T_1 \geq a_d > pT_2$, by (4.10), we know that

$$A_j(t_2^{(d)}) - A_j(T_1) \leq (1+\epsilon)(t_2^{(d)} - T_1) \tag{4.18}$$

By Lemma 5, we have $D_j(t_2^{(d)}) - D_j(T_1) \geq (t_2^{(d)} - T_1) - 5N^3C$. Hence,

$$\begin{aligned} B_j(t_2^{(d)}) &= B_j(T_1) + \big(A_j(t_2^{(d)}) - A_j(T_1)\big) - \big(D_j(t_2^{(d)}) - D_j(T_1)\big) \\ &\leq B_j(T_1) + (1+\epsilon)(t_2^{(d)} - T_1) - \big((t_2^{(d)} - T_1) - 5N^3C\big) \\ &= B_j(T_1) + \epsilon\,(t_2^{(d)} - T_1) + 5N^3C \\ &\leq B_j(T_1) + \epsilon T_2 + 5W^3C \end{aligned} \tag{4.19}$$

Substituting (4.19) into (4.17), we have

$$\begin{aligned} B_j(T_2) &\leq B_j(t_2^{(d)}) + S_{d-1} + 3W^5C \\ &\leq B_j(T_1) + \epsilon T_2 + S_{d-1} + 8W^3C \end{aligned}$$

We need only to consider the following two cases. Both lead to conclusions that contradict (4.12).

(i) If $T_1 > T'$, we have $T_1 \leq T_2 - (t_2^{(d)} - t_1^{(d)}) \leq T_2 - a_d$ and $B_j(T_1) \leq (\gamma+\epsilon)T_1 \leq (\gamma+\epsilon)(T_2 - a_d)$ (due to (4.9)). We have

$$B_j(T_2) \leq B_j(T_1) + \epsilon T_2 + S_{d-1} + 8W^5C$$

$$
\begin{aligned}
&\leq (\gamma+\epsilon)(T_2 - a_d) + \epsilon T_2 + S_{d-1} + 8W^5C \\
&= (\gamma+2\epsilon)T_2 - \big((\gamma+\epsilon)a_d - S_{d-1}\big) + 8W^5C \\
&= (\gamma+2\epsilon)T_2 - (3\epsilon T_2 + 8W^5C) + 8W^5C \\
&= (\gamma-\epsilon)T_2
\end{aligned} \tag{4.20}
$$

The second equality above holds due to Equation (4.14).

(ii) If $T_1 \leq T'$, by (4.3), we have $B_j(T_1) \leq A_j(T_1) \leq NT_1 \leq NT'$ and then

$$
\begin{aligned}
B_j(T_2) &\leq B_j(T_1) + \epsilon T_2 + S_{d-1} + 8W^5C \\
&\leq NT' + \epsilon T_2 + S_{N^2} + 8W^5C \\
&= NT' + \epsilon T_2 + \frac{\gamma-\epsilon}{2}T_2 + 8W^5C \\
&= NT' + \frac{\gamma+\epsilon}{2}T_2 + 8W^5C \\
&\leq (\gamma-\epsilon)T_2
\end{aligned} \tag{4.21}
$$

The last inequality holds because $\epsilon = \min(\frac{\gamma}{4}, \frac{\epsilon'}{2})$, which implies $\gamma - 3\epsilon > 0$, and because $T_2 \geq \frac{2(NT'+8W^5C)}{\gamma-3\epsilon}$.

□

Remark. As mentioned at the beginning of Section 4.3, SRS remains stable when B, H, U queues are not empty to start with (at time 0). More specifically, it can be shown that all results in Sections 4.3.1 and 4.3.2 continue to hold with some minor changes, when initial queue lengths can be nonzero. For example, in that case, we can only claim $G_{jm'}(t_1) \leq G_{jm'}(0)$ in Lemma 5, (instead of $G_{jm'}(t_1) = 0$) so its conclusion has to be changed to $D_j(t_2) - D_j(t_1) \geq (t_2 - t_1) + N(G_{jm'}(t_2) - G_{jm'}(0)) - 5N^3C$.

4.4 Discussions on Stability and Starvation

In this section, we state a few important facts concerning the stability and starvation issues of SRS. We distinguish between two types of such issues. One type, which we believe is of more interest to queueing theory researchers, is whether the length of each bin is growing at most sub-linearly over time (i.e., rate-stable) – and if not, the rate of this linear growth – under various load conditions (admissible or not). The other type, which we believe is of more interest to networking researchers, is whether the length of each bin is small stochastically, under various *admissible* load conditions (e.g., under an offered load of 0.9). In the following, we elaborate on these two types of stability and starvation issues in Sections 4.4.1 and 4.4.2 respectively.

4.4.1 The First Type

We say that a queue Q is rate-starved at (maximum) rate $\bar{\gamma}$ if its queue length $Q(t)$ satisfies $\limsup_{t\to\infty} Q(t)/t = \bar{\gamma} > 0$. Since SRS is proven to be rate-stable (i.e., no bin will have its length grow with a positive rate over time) under mild admissible conditions (see Theorem 2 and the paragraph around Equation (4.2) in Section 4.3), it guarantees no rate-starvation of any bin unless the offered load (the maximum traffic arrival rate to any input or output port) exceeds 1. However, since the traffic arrival rate to any input port cannot exceed 1 (see the paragraph under Inequality (4.3) in Section 4.3), SRS guarantees no rate-starvation unless λ_j, the long-run average traffic arrival rate to an output port j, exceeds 1.

When $\lambda_j > 1$ (for some j), some of the bins whose packets are destined for the output port j will necessarily be rate-starved, no matter what switch scheduling policy is used. However, it can be desirable for the switch to display "grace under fire" in this overload situation in the sense that the switch rate-starves every bin in a fair fashion (e.g., at a rate proportional to the traffic arrival rate to the bin). As it is, an SRS switch cannot guarantee such "grace under fire", as shown in the following example. Suppose an intermediate bin

$H_{i'j'm'}$ has some RSP packets in it when the corresponding input bin $B_{i'j'm'}$ just enters the UFS waiting phase, and from this point onwards, the other $N^2 - 1$ input bins whose packets all destined for the output port j' (namely $\{H_{ij'm}\}_{i,m=1}^{N} \setminus \{H_{i'j'm'}\}$) are all in the UFS evacuation phase and their total traffic arrival rate is exactly 1. In this case, only $B_{i'j'm'}$ is rate-starved (at its traffic arrival rate $\lambda_{i'j'm'}$) and no other input bin in this group is, because the RSP packets in $H_{i'j'm'}$ will never be serviced (as the UFS packets evacuated from the other N^2-1 input bins arrive at the aforementioned intermediate port queue group $\{H_{ij'm}\}_{i,m=1}^{N} \setminus \{H_{i'j'm'}\}$ at rate 1, which is equal to the service rate of this queue group), getting $B_{i'j'm'}$ stuck in the UFS waiting phase.

However, we need only to make the following slight modifications to SRS in order for it achieve a certain degree of "grace under fire" when the switch is only slightly overloaded.

(i) *Upper-bound the length of each H queue.* Whenever the queue length of any H_{ijm} exceeds a certain threshold, H_{ijm} asks B_{ijm} to refrain from sending any more RSP packets over.

(ii) *"Hawking radiation".* All N intermediate ports dedicates 1 switching cycle (N time slots) every T (typically a large constant) cycles, in a *synchronized* manner (otherwise packet reordering could happen), to serving RSP packets, even when there are UFS packets backlogged at the intermediate ports (i.e., U bins are not empty) and/or all H queues are empty.

In other words, we upper-bound the lengths of these H queues (at intermediate ports) and in addition guarantee them (analogous to black holes in the overload situation) a small but constant minimum rate of "Hawking-radiating their mass away". With this modification, SRS can eventually (i.e., in a finite amount of time) clear all H bins whose packets are destined for output port j, thereby allowing all B bins whose packets are destined for output port j to enter the UFS evacuation phase. Again, here and in the following paragraph, this j is chosen arbitrarily, but is fixed once chosen.

The following is true when the output j is only slightly overloaded (i.e., $\lambda_j > 1$ but

barely). From this point onwards (i.e., after all B bins whose packets are destined for output port j have entered the UFS evacuation phase), at each input port, each such B bin is to be serviced at a rate roughly proportional to its traffic arrival rate, since the "UFS orderly evacuation" policy (see Section 4.2.1), which as we have shown guarantees approximate proportional fair rate allocation under admissible traffic, also more roughly does so when the output port j is slightly overloaded. Therefore, each such bin is to be rate-starved at a rate roughly proportional to its traffic arrival rate in the slight overload situation.

The tradeoff however is that the throughput of the switch will be reduced to $1 - 1/T$ although this reduction can be made arbitrarily small by increasing the value of the parameter T. As we will prove in Appendix B.11, the modified policy is strongly stable in the fluid sense, which implies not only rate-stability, but also positive recurrence, when the traffic arrival rate matrix lies in the interior of the reduced capacity region and each element in the traffic arrival matrix process (i.e., the arrival process to each VOQ) is a renewal process[3].

We prefer the unmodified SRS over the modified SRS because the former can provably attain 100% throughput and has good empirical delay performance under *normal* workloads (i.e., when the offered load stays away from 1). We believe the primary mission of a switch scheduling algorithm is to deliver good delay performance under *normal* workloads; such "grace under fire" is a secondary consideration and can be better achieved through other "knobs or levers" orthogonal to switching such as congestion control, packet scheduling, or traffic policing/shaping.

4.4.2 The Second Type

The second type of stability and starvation issues manifest themselves mostly in the empirical delay performance of SRS, which we have studied in Section 4.5 through simulations. Our simulation results, presented in Section 4.5, show that SRS has excellent delay performance under low to moderate traffic loads, which implies at least a good degree of fairness

[3] A similar "Hawking radiation" trick was used, and its stability within the reduced capacity region studied in [52] for a very different application.

in serving different bins and lack of starvation. These simulation studies, however, have not captured the fairness and starvation behaviors of SRS in the heavy-traffic regime, i.e. under traffic loads close to 100%. We acknowledge that, in this heavy-traffic regime, UFS may starve some unfortunate B bins that are waiting on their corresponding H queues to clear at the intermediate ports. These H queues may take a very long time to clear because the intermediate ports prioritize the servicing of UFS packets in U queues over the RSP packets in H queues, and in the heavy traffic regime, these U queues can take a very long time to clear. We emphasize however that, when the switch operates in a persistent heavy-traffic regime, this waiting pain is typically a one-off, because once these H queues are eventually cleared, all bins in the switch will keep operating in the UFS mode for as long as the traffic load is very high, and hence will not face starvation.

4.5 Evaluation

In this section, we compare the performance of our proposed SRS approach, as well as the SRS-UFS variant, with other existing load-balanced switching algorithms, including the basic load-balancing scheme [1], Uniform Frame Spreading (UFS) [5], Full-Ordered Frame First (FOFF) [5], Padded Frames (PF) [4], the Sprinklers scheme [3] and the Concurrent Matching Switch (CMS) [6]. The basic load-balancing scheme (labeled "Basic") does not guarantee packet ordering, but it provides the lower bound of the delay that a load-balanced switch can achieve. UFS, FOFF, PF, Sprinklers and CMS are known to provide reasonably good performance and all of them guarantee packet ordering.

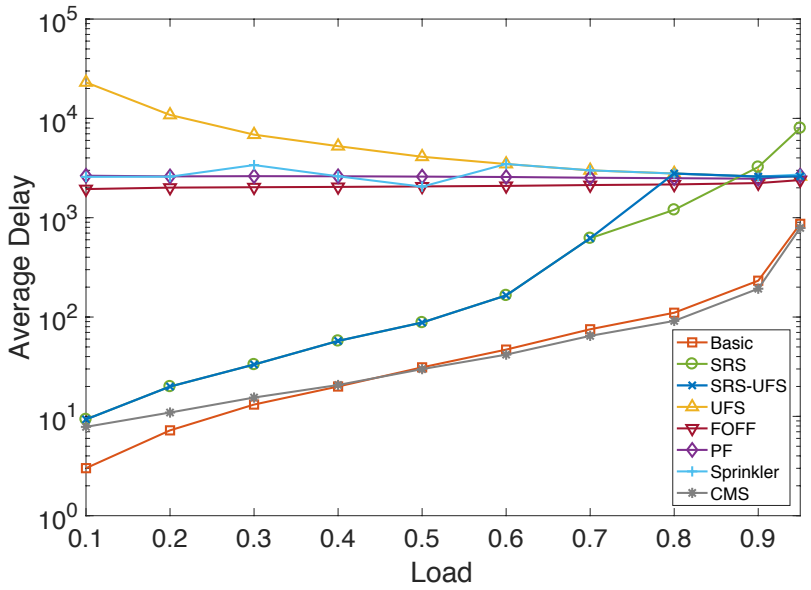
To simulate the effects of grouping application flows (i.e., TCP/UDP flows) into bins by hashing in SRS, we generate application flows over the simulation period using flow statistics measured from real-world Internet traffic traces as follows. We assume that the arrivals of new application flows to an input port follows a Poisson process, and each application flow contains only fixed-length packets, each of which takes exactly 1 time slot to switch. We also assume that the *rate* (in number of fixed-length packets per second)

and the *duration* of each new application flow, viewed as a random vector $\langle \nu, \psi \rangle$, follows the joint empirical distribution measured from the traffic traces. In measuring this rate ν from a packet trace, we "segment" each variable-length packet (whose length information is included in the trace) into fixed-length packets in the sense that we consider a packet of length L (bytes) in the trace $\lceil \frac{L}{500} \rceil$ fixed-length (500-byte-long) packets. The rate of this Poisson process is set according to the intended traffic rate λ of the input port in a simulation run, and the measured empirical average size (in number of fixed-length packets) of an application flow (i.e., $\overline{\nu\psi}$). When a new flow is thus generated with rate $\nu(\omega)$, its traffic arrival process is modeled as i.i.d. Bernoulli in the sense during each time slot, there is a (fixed-length) packet arrival from this flow with probability $\nu(\omega)$.

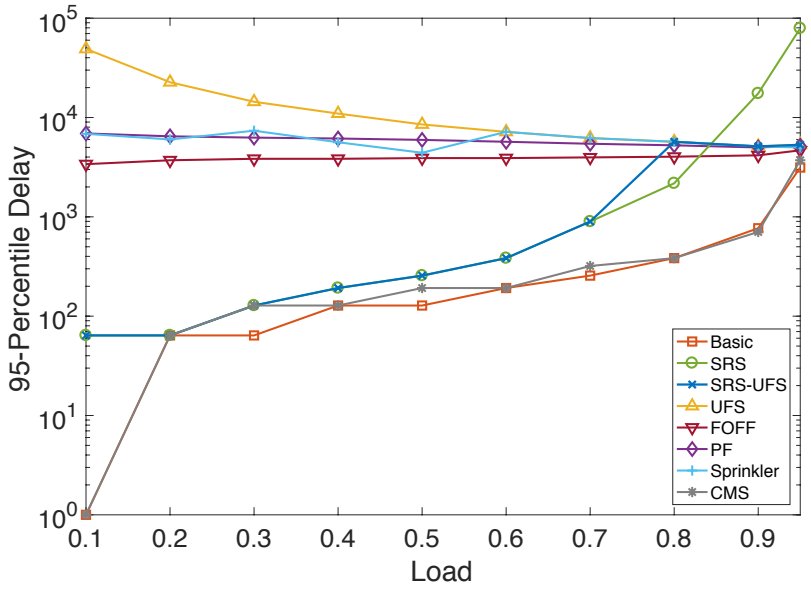
The traces that we used were collected by University of North Carolina (UNC) on a 1 Gbps access link connecting the campus to the rest of the Internet on April 24, 2003. It contains 198,944,706 packet headers and around 13.5 million flows. In our simulation study, traffic into each input port is generated according to the empirical flow statistics measured from this trace. We also use different traffic patterns in our evaluation. The size of the switch in the simulation study is $N = 64$. The RSP-to-UFS mode threshold is set to $W = 2N$ and the initial RSP credit is set to $C = 100N$. For the SRS-UFS variant, the load thresholds that trigger the transitioning between the baseline SRS and the UFS at an input port is set to $\underline{\rho} = 0.75$ and $\bar{\rho} = 0.85$.

Here are the rough guidelines we follow in setting these parameters. First, we make $C \gg W$ to make sure that if the service of a bin is switched from the RSP mode to the UFS mode, it is likely not due to the lack of credit. Second, in setting parameters W and C, we take into consideration the system performance under different traffic loads. Generally speaking, larger W and C values lead to better system performance when traffic load is low to moderate, as such settings keep the system operating mainly in the RSP mode, resulting in a lower delay. However smaller W and C values work better when the traffic load is heavy, as the packets could accumulate rapidly in this case, and should be evacuated in the

UFS mode as soon as possible.



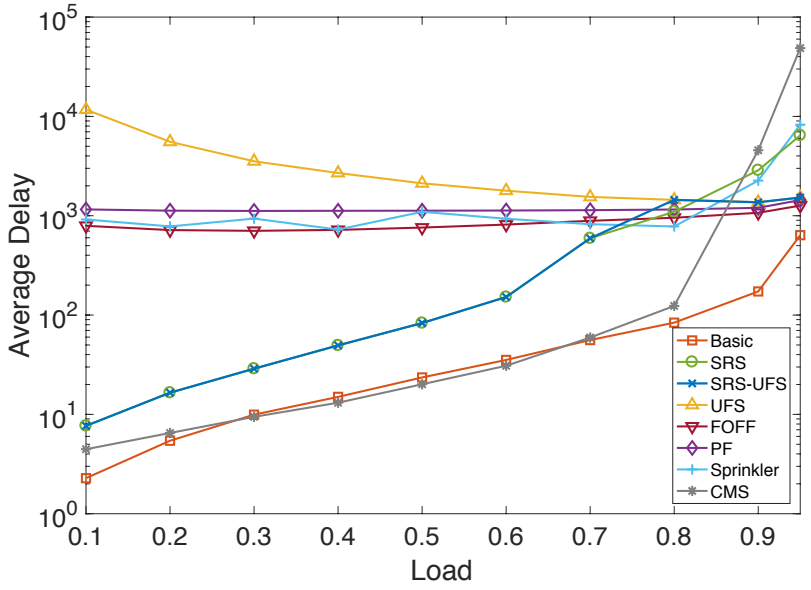
(a) Average delay



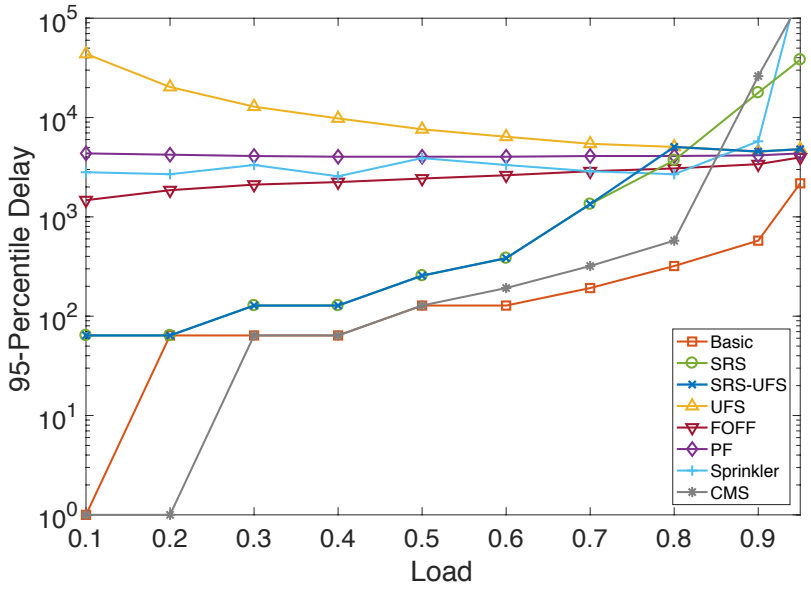
(b) 95-percentile delay

Figure 4.4: Average and 95-percentile delay under uniform traffic.

Our first set of experiments assumes uniform distribution of the destination ports for the arrival flows – i.e. a new flow goes to output j with probability $\frac{1}{N}$. The results are shown in Figure 4.4. The second set of experiments assumes a quasi-diagonal distribution. A new



(a) Average delay



(b) 95-percentile delay

Figure 4.5: Average and 95-percentile delay under quasi-diagonal traffic.

flow arriving at input port i goes to output $j = i$ with probability $\frac{1}{2}$, and goes to any other output port with probability $\frac{1}{2(N-1)}$. The results are shown in Figure 4.5.

The following three observations can be made from the average delay results (Figures 4.4a and 4.5a) of the above experiments. First, for uniform traffic, the average delay

of baseline SRS is significantly better than the existing methods of UFS, FOFF, PF, and Sprinklers for all traffic loads up to about $\lambda < 0.85$. Similarly, for quasi-diagonal traffic, the average delay of baseline SRS is significantly better than the existing methods of UFS, FOFF, PF, and Sprinklers for all traffic loads up to about $\lambda < 0.8$. Second, in comparison to the basic LBS that does not guarantee packet order, the performance of SRS follows roughly the same trend in both experiments. Third, our SRS-UFS variant improves the performance of the baseline SRS at high loads. In both experiments, the SRS-UFS variant scheme performs almost the same as the baseline SRS when the traffic load is low to moderate ($\lambda \leq 0.75$), and as the UFS scheme when traffic load is high ($\lambda > 0.75$).

Similar observations can be made from the 95-percentile delay results shown in Figures 4.4b and 4.5b, except that the degree to which the baseline SRS performs worse than other algorithms under very heavy traffic loads (0.9 and above) in terms of 95-percentile delay is larger than that in terms of average delay. In other words, the baseline SRS has also a larger delay variance than other algorithms under very heavy loads. Our explanation for this phenomenon is as follows. In SRS, since UFS packets take strict priority over RSP packets at both input and intermediate ports, the latter in general have larger delays than the former. The differences between the delays of these two types of packets widen when the traffic load is high, because in this case the RSP packets stuck at an intermediate port can take a long time to clear as they have to "yield" to "passing-by" UFS packets, delaying not only themselves but also packets waiting on them (in the UFS waiting phase) at their respective input ports. As shown in Figures 4.4b and 4.5b, the SRS-UFS variant effectively mitigates this problem, because as described in Section 4.2.3, when the total traffic arrival rate is high, the switch simply serves it using the UFS scheme and hence avoids such long waitings.

As shown in Figures 4.4 and 4.5, CMS has excellent delay performance under low-to-moderate loads. However, this comes with high communication and implementation costs, as compared to SRS and other LBS schemes. In terms of the communication cost, CMS

requires, for each packet transmission, the exchange of two token messages. In terms of the implementation cost, CMS requires the computation of a matching between the input ports and the output ports, every N time slots. Although in CMS, the complexity of this computation is amortized (over N time slots) to $O(1)$ per time slot, using the SERENA matching algorithm [53], which has $O(N)$ complexity, the need to implement, and to have a fast processor execute, SERENA undoubtedly adds to the implementation cost of the switch. In comparison, no other LBS solution requires any matching computation. Finally, the delay performance of CMS is poor under high loads (≥ 0.9) for non-uniform traffic, as shown in Figure 4.5.

4.6 Conclusions

In this chapter, we proposed SRS, a simple randomized load-balanced switch architecture based on the hashing of application flows, combined with safety mechanisms to prevent unstable build-up of packets at intermediate queues throughout the switch. It has the lowest possible computational complexity ($O(1)$ per packet per port) and is easy to implement. We rigorously proved that the proposed SRS guarantees both stability under arbitrary admissible traffic and packet ordering. We showed experimentally that SRS has competitive delay performance compared to other solutions.

CHAPTER 5
CONCLUSIONS

This thesis proposed several load-balanced switch architectures that can maintain packet ordering while still achieve low latency and provide throughput guarantees.

In Chapter 3, we show that the amount of packet reordering that can occur with the load-balanced switch is actually quite limited, which means that packet reordering can simply be rectified by employing reordering buffers at the switch outputs. In particular, we formally bound the worst-case amount of time that a packet has to wait in these output reordering buffers before it is guaranteed to be ready for in-order departure with high probability, and we prove that this bound is *linear* with respect to the switch size. This linear bound is significant because previous approaches can add quadratic or cubic delays to the load-balanced switch. In addition, we use a hash-grouping method that further reduces resequencing delays significantly.

In Chapter 4, we proposed SRS, a simple randomized load-balanced switch architecture based on the hashing of application flows, combined with safety mechanisms to prevent unstable build-up of packets at intermediate queues throughout the switch. It has the lowest possible computational complexity ($O(1)$ per packet per port) and is easy to implement. We rigorously proved that the proposed SRS guarantees both stability under arbitrary admissible traffic and packet ordering. We believe that this work will serve as a catalyst to a rich family of solutions based on the simple principles of flow randomization.

Although simple and intuitive, our experimental results show that our output packet resequencing approach and our extended randomized load-balancing approach outperforms existing load-balanced switch architectures.

Appendices

APPENDIX A

APPENDIX FOR CHAPTER 3

A.1 Proof of Theorem 1

The proof is a straightforward algebraic evaluation. For $k > d$, we have

$$
\begin{aligned}
&P(L_1 - L_2 = k) = \sum_{i=0}^{\infty} P(L_2 = i) P(L_1 = i + k) \\
&= \sum_{i=0}^{\infty} \pi_i \pi_{i+k} \\
&= \sum_{i=0}^{d} \pi_i \pi_{i+k} + \sum_{i=d+1}^{\infty} \pi_i \pi_{i+k} \\
&\leq \sum_{i=0}^{d} \pi_i \overline{\pi}_{i+k} + \sum_{i=d+1}^{\infty} \overline{\pi}_i \overline{\pi}_{i+k} \\
&\leq \sum_{i=0}^{d} \pi_i \left(-\frac{\alpha_0}{\zeta_0^{i+k+1}} + \frac{M_1(r)}{r^{i+k}} \right) \\
&\quad + \sum_{i=d+1}^{\infty} \left(-\frac{\alpha_0}{\zeta_0^{i+1}} + \frac{M_1(r)}{r^{i}} \right) \left(-\frac{\alpha_0}{\zeta_0^{i+k+1}} + \frac{M_1(r)}{r^{i+k}} \right) \\
&= -\left(\sum_{i=0}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{i+1}} \right) \frac{1}{\zeta_0^k} + \left(\sum_{i=0}^{d} \frac{\pi_i M_1(r)}{r^i} \right) \frac{1}{r^k} \\
&\quad + \sum_{i=d+1}^{\infty} \left(\frac{\alpha_0^2}{\zeta_0^{k+2}} \frac{1}{\zeta_0^{2i}} - \frac{\alpha_0 M_1(r)}{\zeta_0 r^k} \frac{1}{\zeta_0^i r^i} - \frac{\alpha_0 M_1(r)}{\zeta_0^{k+1}} \frac{1}{\zeta_0^i r^i} + \frac{M_1^2(r)}{r^k} \frac{1}{r^{2i}} \right) \\
&= -\left(\sum_{i=0}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{i+1}} \right) \frac{1}{\zeta_0^k} + \left(\sum_{i=0}^{d} \frac{\pi_i M_1(r)}{r^i} \right) \frac{1}{r^k} \\
&\quad + \frac{\alpha_0^2}{\zeta_0^{k+2}} \sum_{i=d+1}^{\infty} \frac{1}{\zeta_0^{2i}} - \frac{\alpha_0 M_1(r)}{\zeta_0 r^k} \sum_{i=d+1}^{\infty} \frac{1}{\zeta_0^i r^i} - \frac{\alpha_0 M_1(r)}{\zeta_0^{k+1}} \sum_{i=d+1}^{\infty} \frac{1}{\zeta_0^i r^i} + \frac{M_1^2(r)}{r^k} \sum_{i=d+1}^{\infty} \frac{1}{r^{2i}} \\
&= -\left(\sum_{i=0}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{i+1}} \right) \frac{1}{\zeta_0^k} + \left(\sum_{i=0}^{d} \frac{\pi_i M_1(r)}{r^i} \right) \frac{1}{r^k} \\
&\quad + \frac{\alpha_0^2}{\zeta_0^{k+2}} \frac{1}{\zeta_0^{2(d+1)}(1 - \zeta_0^{-2})} - \frac{\alpha_0 M_1(r)}{\zeta_0 r^k} \frac{1}{\zeta_0^{d+1} r^{d+1} (1 - \zeta_0^{-1} r^{-1})}
\end{aligned}
$$

$$
\begin{aligned}
&- \frac{\alpha_0 M_1(r)}{\zeta_0^{k+1}} \frac{1}{\zeta_0^{d+1} r^{d+1}(1-\zeta_0^{-1}r^{-1})} + \frac{M_1^2(r)}{r^k} \frac{1}{r^{2(d+1)}(1-r^{-2})} \\
&= -\left(\sum_{i=0}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{i+1}} \right) \frac{1}{\zeta_0^k} + \left(\sum_{i=0}^{d} \frac{\pi_i M_1(r)}{r^i} \right) \frac{1}{r^k}, \\
&+ \frac{\alpha_0^2}{\zeta_0^{2d+4}(1-\zeta_0^{-2})} \frac{1}{\zeta_0^k} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1}r^{-1})} \frac{1}{r^k} \\
&- \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1}r^{-1})} \frac{1}{\zeta_0^k} + \frac{M_1^2(r)}{r^{2d+2}(1-r^{-2})} \frac{1}{r^k} \\
&= \left(-\sum_{i=0}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{i+1}} + \frac{\alpha_0^2}{\zeta_0^{2d+4}(1-\zeta_0^{-2})} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1}r^{-1})} \right) \frac{1}{\zeta_0^k} \\
&+ \left(\sum_{i=0}^{d} \frac{\pi_i M_1(r)}{r^i} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1}r^{-1})} + \frac{M_1^2(r)}{r^{2d+2}(1-r^{-2})} \right) \frac{1}{r^k}
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
&P(L_1 - L_2 \geq \theta) \\
&= \sum_{k=\theta}^{\infty} P(L_1 - L_2 = k) \\
&\leq \left(-\sum_{i=0}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{i+1}} + \frac{\alpha_0^2}{\zeta_0^{2d+4}(1-\zeta_0^{-2})} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1}r^{-1})} \right) \sum_{k=\theta}^{\infty} \frac{1}{\zeta_0^k} \\
&\quad + \left(\sum_{i=0}^{d} \frac{\pi_i M_1(r)}{r^i} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1}r^{-1})} + \frac{M_1^2(r)}{r^{2d+2}(1-r^{-2})} \right) \sum_{k=\theta}^{\infty} \frac{1}{r^k} \\
&= \left(-\sum_{i=0}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{i+1}} + \frac{\alpha_0^2}{\zeta_0^{2d+4}(1-\zeta_0^{-2})} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1}r^{-1})} \right) \frac{1}{(1-\zeta_0^{-1})} \frac{1}{\zeta_0^\theta} \\
&\quad + \left(\sum_{i=0}^{d} \frac{\pi_i M_1(r)}{r^i} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1}r^{-1})} + \frac{M_1^2(r)}{r^{2d+2}(1-r^{-2})} \right) \frac{1}{(1-r^{-1})} \frac{1}{r^\theta} \\
&= C_\zeta^{(d)} \frac{1}{\zeta_0^\theta} + C_r^{(d)} \frac{1}{r^\theta}
\end{aligned}
$$

which is (3.5) in Theorem 1.

Similarly, for $k \leq d$, we have

$$
P(L_1 - L_2 = k) = \sum_{i=0}^{\infty} P(L_2 = i) P(L_1 = i + k)
$$

$$
\begin{aligned}
&= \sum_{i=0}^{\infty} \pi_i \pi_{i+k} \\
&= \sum_{i=0}^{d-k} \pi_i \pi_{i+k} + \sum_{i=d-k+1}^{d} \pi_i \pi_{i+k} + \sum_{i=d+1}^{\infty} \pi_i \pi_{i+k} \\
&\leq \sum_{i=0}^{d-k} \pi_i \pi_{i+k} + \sum_{i=d-k+1}^{d} \pi_i \overline{\pi}_{i+k} + \sum_{i=d+1}^{\infty} \overline{\pi}_i \overline{\pi}_{i+k} \\
&= \sum_{i=0}^{d-k} \pi_i \pi_{i+k} + \sum_{i=d-k+1}^{d} \pi_i \left(-\frac{\alpha_0}{\zeta_0^{i+k+1}} + \frac{M_1(r)}{r^{i+k}} \right) \\
&\quad + \sum_{i=d+1}^{\infty} \left(-\frac{\alpha_0}{\zeta_0^{i+1}} + \frac{M_1(r)}{r^i} \right) \left(-\frac{\alpha_0}{\zeta_0^{i+k+1}} + \frac{M_1(r)}{r^{i+k}} \right) \\
&= \sum_{i=0}^{d-k} \pi_i \pi_{i+k} - \left(\sum_{i=d-k+1}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{i+1}} \right) \frac{1}{\zeta_0^k} + \left(\sum_{i=d-k+1}^{d} \frac{\pi_i M_1(r)}{r^i} \right) \frac{1}{r^k} \\
&\quad + \frac{\alpha_0^2}{\zeta_0^{2d+4}(1-\zeta_0^{-2})} \frac{1}{\zeta_0^k} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1} r^{-1})} \frac{1}{\zeta_0^k} \\
&\quad - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1} r^{-1})} \frac{1}{r^k} + \frac{M_1^2(r)}{r^{2d+2}(1-r^{-2})} \frac{1}{r^k} \\
&= \sum_{i=0}^{d-k} \pi_i \pi_{i+k} + \left(-\sum_{i=d-k+1}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{i+1}} + \frac{\alpha_0^2}{\zeta_0^{2d+4}(1-\zeta_0^{-2})} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1} r^{-1})} \right) \frac{1}{\zeta_0^k} \\
&\quad + \left(\sum_{i=d-k+1}^{d} \frac{\pi_i M_1(r)}{r^i} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1} r^{-1})} + \frac{M_1^2(r)}{r^{2d+2}(1-r^{-2})} \right) \frac{1}{r^k}
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
&P(L_1 - L_2 \geq \theta) \\
&= \sum_{k=\theta}^{d} P(L_1 - L_2 = k) + P(L_1 - L_2 \geq d+1) \\
&\leq \sum_{k=\theta}^{d} P(L_1 - L_2 = k) + \left(C_\zeta^{(d)} \frac{1}{\zeta_0^{d+1}} + C_r^{(d)} \frac{1}{r^{d+1}} \right) \\
&\leq \sum_{k=\theta}^{d} \sum_{i=0}^{d-k} \pi_i \pi_{i+k} + \sum_{k=\theta}^{d} \left(-\sum_{i=d-k+1}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{i+1}} + \frac{\alpha_0^2}{\zeta_0^{2d+4}(1-\zeta_0^{-2})} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1} r^{-1})} \right) \frac{1}{\zeta_0^k} \\
&\quad + \sum_{k=\theta}^{d} \left(\sum_{i=d-k+1}^{d} \frac{\pi_i M_1(r)}{r^i} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1} r^{-1})} + \frac{M_1^2(r)}{r^{2d+2}(1-r^{-2})} \right) \frac{1}{r^k}
\end{aligned}
$$

$$+\left(C_{\zeta}^{(d)}\frac{1}{\zeta_0^{d+1}}+C_r^{(d)}\frac{1}{r^{d+1}}\right)$$
$$=\sum_{k=\theta}^{d}\sum_{i=0}^{d-k}\pi_i\pi_{i+k}+\widetilde{C}_{\zeta}^{(\theta,d)}+\widetilde{C}_r^{(\theta,d)}+C_{\zeta}^{(d)}\frac{1}{\zeta_0^{d+1}}+C_r^{(d)}\frac{1}{r^{d+1}}$$

which is (3.6) in Theorem 1.

APPENDIX B

APPENDIX FOR CHAPTER 4

B.1 Service rate limit of input port bins under RSP mode

In this section, we justify the following statement that we made earlier, which serves as the rationale for limiting the rate, at which each bin B_{ijm} can receive switching service under the RSP mode, to $\frac{\lambda_{ij}}{N}$.

> For any $1 \leq i, j, m \leq N$, $\frac{\lambda_{ij}}{N}$ is the highest RSP traffic rate the input port i can grant, under any "nondiscriminatory policy" (i.e., with a policy statement that does not "discriminate against" any particular values of i, j, m), to the bin B_{ijm} without risking compromising the (rate) stability of the set of bins that buffer packets destined for the output port j, at the intermediate port m.

Since it has been proven in Theorem 2 that the rate limit $\frac{\lambda_{ij}}{N}$ is safe (i.e., it will not compromise the rate stability of the switch), we need only to show here that this rate limit cannot be exceeded, which we "prove by contradiction" as follows. We come up with a set of traffic arrival rates (to the bins) that are admissible, and show that a certain set of queues at the intermediate port m have a larger total packet arrival rate than their total departure rate, and hence are not rate-stable. Now fix the values of j and m and let θ_{ijm} be the (upper) rate limit under which traffic in bin B_{ijm} can be served under the RSP mode, for $i = 1, 2, \ldots, N$. Let the arrival rates (to VOQ_{ij}) λ_{ij}, $j = 1, 2, \ldots, N$ satisfy $\sum_{i=1}^{N} \lambda_{ij} = 1$; this set of rates is clearly admissible (for rate-stability).

Suppose, for any $1 \leq i \leq N$, the RSP rate limit imposed on each bin B_{ijm}, satisfies $\theta_{ijm} > \frac{\lambda_{ij}}{N}$, for $i = 1, 2, \ldots, N$. We set each λ_{ijm} to $\min\{\theta_{ijm}, \lambda_{ij}\}$, for $i = 1, 2, \ldots, N$. In other words, we assume each input port i TCP-hashes its incoming traffic in the amount of exactly θ_{ijm} to bin B_{ijm}. This assumption is valid for this "proof by contradiction", because

this situation could in theory happen (e.g., when, at each input port i, VOQ_{ij} contained a long-lived elephant flow that is TCP-hashed to bin B_{ijm}). In this situation, the total amount of RSP traffic that arrives at the intermediate port m and is destined for intermediate port j is equal to $\sum_{i=1}^{N} \lambda_{ijm} > \sum_{i=1}^{N} \frac{\lambda_{ij}}{N} = \frac{1}{N}$. In other words, the total arrive rate to the set of bins that buffer packets destined for the output port j at the intermediate port m is larger than $\frac{1}{N}$. However, these bins are serviced at rate $\frac{1}{N}$, as the second switching fabric connects intermediate port m with output port j once every N time slots, and hence are not (rate) stable.

Readers may wonder that to avoid this overload situation, some input port bins can be allowed to exceed this rate limit while others are not. However, the resulting rate-limit policy is no longer "nondiscriminatory" since it "discriminates" against the latter set of bins.

B.2 UFS orderly evacuation scheduling scheme

Algorithm 4 Scheduling UFS packets at input port i

1: **Initialize:** Set all UFS pressure counters to $C_{ijm}^{U} = 0, j, m = 1, 2, \ldots, N$;

Upon the arrival of a packet to B_{ijm}:

2: **if** B_{ijm} is in UFS evacuation phase **then**
3: $\quad C_{ijm}^{U} \leftarrow C_{ijm}^{U} + 1$;

Upon the departure of a packet from B_{ijm}:

4: **if** B_{ijm} is in UFS evacuation phase **then**
5: $\quad C_{ijm}^{U} \leftarrow \max\left(0, C_{ijm}^{U} - 1\right)$;

If there are multiple USF-ready bin waiting to be served:

6: Pick the UFS-ready bin B_{ijm} with the largest UFS pressure counter value to serve in the next N time slots;

To achieve the goal of orderly evacuation among UFS-ready bins, we piece together a scheduler that is low in both computational and implementation complexities, using "off-the-shelf" rate control mechanisms. Its pseudocode is shown in Algorithm 4.

As shown in Algorithm 4, each bin B_{ijm} is associated with a UFS "pressure" counter C^U_{ijm} which is set to 0 (Line 1 in Algorithm 4) when B_{ijm} becomes UFS-ready. This counter C^U_{ijm} tracks, since the last time it was set or decreased to 0, whether the queue length of B_{ijm} has increased – due to the number of new packet arrivals ("pressure build-up") exceeding the number of B_{ijm} packets that has been served ("pressure relief") – and if so, by how much, as follows. This pressure counter remains 0 when B_{ijm} is in the UFS waiting phase. After B_{ijm} enters the UFS evacuation phase, we increment C^U_{ijm} by 1 with each new packet arrival to B_{ijm} (Line 3). When a frame of N packets depart from a UFS-ready bin B_{ijm}, we decrement C^U_{ijm} by N. If the value of C^U_{ijm} becomes negative after this decrement, we reset it to 0. This is equivalent to executing Lines 4 - 5 in Algorithm 4 N times. We purposefully write the pseudo code this way to make it consistent with the proof of Lemma 7.

As is clear from our orderly evacuation goal, the larger the value of C^U_{ijm} is, the more urgently B_{ijm} should be served. Hence our scheduler adopts the following simple service discipline: Among all UFS-ready bins, the one with the largest UFS pressure counter value (with ties broken arbitrarily) is served next (Line 6). To implement this selection policy, we need only to maintain a heap of UFS-ready bins indexed by pressure counter values. Although the complexity of this implementation is $O(\log N)$, this complexity is not a major concern because, as we will show in Appendix B.3, our scheduler only incur this complexity once per frame (of N packets), and hence the amortized complexity (per packet per port) is just $O(1)$.

B.3 Implementation and Complexity

In this section, we discuss in details how to efficiently implement the SRS operations described in Section 4.2 and Appendix B.2 so that their total computational complexity is $O(1)$ per packet per port.

The first issue, mentioned in Section 4.2.1, is that RSP-ready bins among bins B_{i1m},

B_{i2m}, ..., B_{iNm} at input port i (those in the "m^{th} row" highlighted in Figure 4.1) should receive RSP service (i.e., to have an HOL packet switched to intermediate port m) in the round-robin order. Suppose, among these RSP-ready bins, B_{ijm} (where the "m^{th} row" intersects "j^{th} column" in Figure 4.1) was the last to receive RSP service. Clearly, our computation task here is to locate the next (per the round-robin order) RSP-ready bin in this "m^{th} row" when needed. A naive implementation of this computation task is to "linearly scan", starting from $B_{i[j+1]m}$, the "m^{th} row" for the next RSP-ready bin. This naive implementation, however, could incur a high computational complexity of $O(N)$ per packet per port in the worst-case, e.g., when B_{ijm} is the only RSP-ready bin in this "m^{th} row". A better approach, which has a computational complexity of only $O(1)$ per packet per port, is to maintain a linked list of non-empty RSP-ready bins. A bin is inserted to the linked list when it becomes both RSP-ready (say after receiving a unit of credit due to the above-mentioned round-robin credit distribution) and non-empty, and is deleted from the linked list when it is no longer both RSP-ready and non-empty. In addition, a pointer is associated with this linked list to remember (i.e., to point to) the RSP-ready bin to be served next. To the best of our knowledge, such a linked list implementation of efficient round-robin rotation among a dynamic set of entities can be traced as far back as to the Deficit Round Robin (DRR) packet scheduler [50].

The second implementation issue, mentioned in Section 4.2.1, is how to avoid the high computational complexity of $O(N)$ per packet per port involved in distributing credits (Line 3 of Algorithm 3), in the amount of $\frac{1}{N}$ each, to all N credit counters $C^R_{ij1}, C^R_{ij2}, \ldots, C^R_{ijn}$ (those in the "j^{th} column" highlighted in Figure 4.1). Our solution is to increment, instead of all these N counters by $\frac{1}{N}$ each, a single counter by 1 in the round-robin manner, thereby reducing the computational complexity to $O(1)$ per packet per port. In other words, if we last incremented credit counter C^R_{ijm}, then we should increment counter $C^R_{ij[m+1]}$ this time. To implement this round-robin rotation, we simply need to remember, for each such a set of counters $C^R_{ij1}, C^R_{ij2}, \ldots, C^R_{ijN}$, the index of the counter that was last incremented, using

a pointer p_{ij}. This round-robin rotation distributes credits almost as evenly as the original scheme of incrementing all N counters: It can be shown that Lemma 1 continues to hold with a slightly larger constant bound ($NC + 1$ instead of NC). Our stability proof, which assumes the bound NC, only needs to be slightly modified to accommodate the slightly larger bound $NC + 1$.

The third issue, mentioned in Section 4.2.1 and Appendix B.2, is how to efficiently implement Algorithm 4 so that its computational complexity is only $O(1)$ per packet per port. With the standard data structure of organizing the pressure counters as a heap keyed by their values, the time complexity of Algorithm 4 is $O(\log N)$ per packet per port. Our solution is to associate another counter, called lazy pressure counter, with each bin. We increment the value of a lazy pressure counter only once by N after every N increments to the corresponding normal pressure counter. Hence, on average only one lazy pressure counter is incremented (by N) every N time slots. In our solution, the heap data structure is keyed instead by the values of lazy pressure counters. This way, on average only one heapify operation, which has a time complexity of $O(\log N)$, is triggered every frame (i.e., N time slots).[1] It can be shown that, by making scheduling decisions based on the slightly outdated "pressure readings" from lazy pressure counters, our streamlined scheduler increases the aforementioned fluctuation bound N^3 only slightly. Our stability proof, based on the guarantees of the original scheduler, only needs to be slightly modified to accommodate this increase.

The fourth issue, mentioned in Section 4.2.2 in a footnote, is how to efficiently schedule those intermediate bins among $H_{1jm}, H_{2jm}, \ldots, H_{Njm}$ that are non-empty (i.e., have at least one packet in the bin), in the round-robing manner. This issue is almost identical to the first issue, except that there is no RSP credit payment and redistribution to worry about (so it is even easier to solve). Hence the solution to the second issue, which has a computational

[1] Although in the worst case there can be a burst of N^2 consecutive increments to the lazy counters, using a (counter) "seed value" randomization technique introduced in [54], we can ensure that, with overwhelming probability, at most $O(1)$ increments to lazy pressure counters can be triggered during every N consecutive time slots.

complexity of $O(1)$ per packet per port, applies to this one.

The last issue is about the extra space complexity needed for SRS to maintain N^2 logical bins, instead of N VOQs, at each input port. This extra space complexity is relatively modest for the following reason. Like a VOQ in a standard input-queued crossbar switch, each bin here is typically implemented as a linked list of nodes, each of which contains a *pointer* to the memory address in the packet buffer space where the actual packet is stored. With such an implementation, at each input port, the only difference in memory requirement between an SRS and a standard input-queued crossbar switch, is that the former requires N^2 head pointers, one for each bin that points to (the first node of) the corresponding linked list, whereas the latter requires only N of them, one for each VOQ. Since each pointer requires only 4 bytes to store whereas each packet can be hundreds or thousands of bytes long, the extra memory space needed to store $N^2 - N$ more pointers is typically relatively modest compared to that needed to store the actual packets.

B.4 Extremely bursty process can still have long-run average rate

In this section, we show that an arrival process could be extremely bursty yet still has a long-run average rate. To do so, we introduce the notion of on-off process: In such a process, during each time slot, we say that the process is "on" if there is a packet arrival, and is "off" otherwise. Now consider the following on-off process. It is on for 1 time slot, off for 1 time slot, on for 2 time slots, off for 2 time slots, on for 3 time slots, off for 3 time slots, and so on forever. In other words, the process is on and then off each for an interval that is increasing linearly over time. Clearly, this process is very bursty because, given any (arbitrarily large) time duration τ, we can find an interval of length τ (time slots) in which the average arrival rate of the process is strictly 0 (i.e, no arrival at all during the interval) and another interval of length τ in which the average arrival rate of the process is strictly 1 (i.e., τ arrivals during the interval). It is not hard to verify, however, that the long-run average arrival rate of this arrival process converges to $\lambda = 0.5$.

B.5 Proof of Lemma 1

Proof. Let $C^R_{ijm}(t)$ and $C^R_{ijm'}(t)$ denote the values of credit counters C^R_{ijm} and $C^R_{ijm'}$ at time slot t respectively. Note that for any input port queue group B_{ij}, the total RSP credits stay unchanged at anytime. So we always have $C^R_{ijm}(t) + C^R_{ijm'}(t) \leq NC$.

Let $D^R_{ij}(t) \equiv \sum_{m=1}^{N} D^R_{ijm}(t)$ be the cumulative number of packet departures from queue group B_{ij} in RSP mode by time slot t. From Lines 2 - 3 in Algorithm 3, we know that

$$C^R_{ijm}(t) = C - D^R_{ijm}(t) + \frac{1}{N} D^R_{ij}(t)$$

which is equivalent to $D^R_{ijm}(t) = C + \frac{1}{N} D^R_{ij}(t) - C^R_{ijm}(t)$. Similarly, we have $D^R_{ijm'}(t) = C + \frac{1}{N} D^R_{ij}(t) - C^R_{ijm'}(t)$. Thus, we have

$$\begin{aligned} |D^R_{ijm}(t) - D^R_{ijm'}(t)| &= |C^R_{ijm}(t) - C^R_{ijm'}(t)| \\ &\leq |C^R_{ijm}(t) + C^R_{ijm'}(t)| \\ &\leq NC \end{aligned}$$

□

B.6 Proof of Lemma 4

Proof. As mentioned earlier, the set of $I_{jm}(t)$ (defined in Section 4.3.1) packets that arrive at the queue group G_{jm} during the time interval $[0, t]$ can be classified into two types: those RSP packets sent to $H_{1jm}, H_{2jm}, \ldots, H_{Njm}$ and those UFS packets sent to U_{jm} both from input ports. We denote the total counts of these two types of packets by $I^R_{jm}(t)$ and $I^U_{jm}(t)$, respectively. Let $I^R_{ijm}(t)$ be the number of (RSP) packets that arrive at H_{ijm} during the time interval $[0, t]$. We have $I^R_{jm}(t) = \sum_{i=1}^{N} I^R_{ijm}(t)$, $j, m = 1, \ldots, N$ by definition. We also have $I^R_{ijm}(t) = D^R_{ijm}(t)$, $i, j, m = 1, \ldots, N$, because every RSP packet that departs from

B_{ijm} by time t arrives at H_{ijm} by time t. Therefore, we have

$$\begin{aligned} I^R_{jm}(t) - I^R_{jm'}(t) &= \sum_{i=1}^{N} \left(I^R_{ijm}(t) - I^R_{ijm'}(t) \right) \\ &= \sum_{i=1}^{N} \left(D^R_{ijm}(t) - D^R_{ijm'}(t) \right) \end{aligned}$$

By Lemma 1, we have

$$\left| I^R_{jm}(t) - I^R_{jm'}(t) \right| \leq \sum_{i=1}^{N} \left| D^R_{ijm}(t) - D^R_{ijm'}(t) \right| \leq N^2 C$$

Furthermore, if an input port bin is in UFS mode, around the time it sends one packet to intermediate port m, it must also send one packet from the same frame to intermediate port m' within the same cycle (N time slots). Thus, we always have $\left| I^U_{jm}(t) - I^U_{jm'}(t) \right| \leq 1$ for any $t \geq 0$. Therefore,

$$\begin{aligned} |I_{jm}(t) - I_{jm'}(t)| &= \left| (I^R_{jm}(t) + I^U_{jm}(t)) - (I^R_{jm'}(t) + I^U_{jm'}(t)) \right| \\ &\leq \left| I^R_{jm}(t) - I^R_{jm'}(t) \right| + \left| I^U_{jm}(t) - I^U_{jm'}(t) \right| \\ &= N^2 C + 1 \end{aligned}$$

□

B.7 Proof of Lemma 5

Proof. Let $O_{jm}(t)$ be the cumulative number of packets departure from queue group G_{jm} by time slot t. Since G_{jm} is non-empty from time slot $(t_1 + 1)$ to t_2, it must send out 1 packet per N time slots. We have

$$\begin{aligned} & I_{jm}(t_2) - I_{jm}(t_1) \\ &= O_{jm}(t_2) - O_{jm}(t_1) + G_{jm}(t_2) - G_{jm}(t_1) \end{aligned}$$

$$\geq \left\lfloor \frac{1}{N}(t_2 - t_1) \right\rfloor + G_{jm}(t_2)$$
$$\geq \frac{1}{N}(t_2 - t_1) - 1 + G_{jm}(t_2)$$

where $\lfloor x \rfloor$ is the largest integer smaller than or equal to x. From Lemma 4, for any intermediate port $m' = 1, \ldots, N$, we always have $I_{jm'}(t_2) \geq I_{jm}(t_2) - (N^2C + 1)$ and $I_{jm'}(t_1) \leq I_{jm}(t_1) + N^2C + 1$ (which will be used in the first inequality below). Note that $D_j(t) = \sum_{m'=1}^{N} I_{jm'}(t)$. We have

$$\begin{aligned}
D_j(t_2) - D_j(t_1) &= \sum_{m'=1}^{N} I_{jm'}(t_2) - \sum_{m'=1}^{N} I_{jm'}(t_1) \\
&= \sum_{m'=1}^{N} \big(I_{jm'}(t_2) - I_{jm'}(t_1)\big) \\
&\geq \sum_{m'=1}^{N} \big(I_{jm}(t_2) - (N^2C + 1) - \big(I_{jm}(t_1) + N^2C + 1\big)\big) \\
&= \sum_{m'=1}^{N} \big(I_{jm}(t_2) - I_{jm}(t_1) - 2(N^2C + 1)\big) \\
&\geq \sum_{m'=1}^{N} \left(\frac{1}{N}(t_2 - t_1) - 1 + G_{jm}(t_2) - 2(N^2C + 1)\right) \\
&\geq (t_2 - t_1) + NG_{jm}(t_2) - 5N^3C
\end{aligned}$$

□

B.8 Proof of Lemma 2

For convenience of presentation, we simply assume that B_{ijm} experiences a (degenerated) UFS waiting period of length 0 if H_{ijm} is already cleared (i.e., has length 0) when B_{ijm} enters the UFS mode. Recall from Appendix B.2 that C^U_{ijm} is the UFS pressure counter associated with bin B_{ijm}. Let $C^U_{ijm}(t)$ denote the value of C^U_{ijm} at time t. As mentioned earlier, when B_{ijm} is in the UFS evacuation phase, C^U_{ijm} tracks the change of its queue

length except that it could be allowed to transmit a UFS packet (as a part of a UFS frame) at time t, even if $C^U_{ijm}(t)$ is 0 (Line 5 in Algorithm 4), and $C^U_{ijm}(t)$ remains 0 after such a transmission. We say B_{ijm} "steals service" at such moments (without having $C^U_{ijm}(t)$ decremented). Our scheduler purposefully allows such a behavior, since Algorithm 4 guarantees that B_{ijm} needs UFS evacuation most urgently whenever it is scheduled (Line 6). If B_{ijm} steals service at time t, UFS pressure counters of all other UFS-ready bins at input port i should also have values equal or close to 0 at that time, indicating none of them needs evacuation urgently anyway. Hence B_{ijm} should not be punished for stealing service at time t. To prove Lemma 2, we need the following technical lemma.

Lemma 7. *If none of the bins at input port i has ever stolen service throughout time slots t_1 to t_2, we must have*

$$\sum_{j,m=1}^{N} C^U_{ijm}(t_2) \leq \max\left(N, \sum_{j,m=1}^{N} C^U_{ijm}(t_1)\right)$$

Proof. Let $\mathcal{B}^U_i(t)$ be the set of bins at input port i that are in the UFS evacuation phase at time t. If $\mathcal{B}^U_i(t_2) = \emptyset$, we must have $C^U_{ijm}(t_2) = 0$ for $j, m = 1, 2, \ldots, N$ and thus $\sum_{j,m=1}^{N} C^U_{ijm}(t_2) = 0 \leq N$. Otherwise, let $t' \in [0, t_2]$ be the (unique) time such that $\mathcal{B}^U_i(t') = \emptyset$ and $\mathcal{B}^U_i(t) \neq \emptyset$ for any time slot $t \in [t'+1,\, t_2]$. Note that t' must exist since $\mathcal{B}^U_i(0) = \emptyset$. Then we must have $\sum_{j,m=1}^{N} C^U_{ijm}(t') = 0$. Note that when input port i is connected to intermediate port 1 at a time slot $t'' \in [t'+1,\, t'+N]$, one of the following statements must be true.

- If $t_2 \leq t''$, we have $t_2 - t' \leq N$. Note that, as assumed in Section 4.3, at most one packet can arrive at each input port in a single time slot and the value of $\sum_{j,m=1}^{N} C^U_{ijm}$ is incremented by at most 1 per packet arrival (Line 3 in Algorithm 4), we can obtain

$$\sum_{j,m=1}^{N} C^U_{ijm}(t_2) \leq \sum_{j,m=1}^{N} C^U_{ijm}(t') + (t_2 - t') \leq N$$

- If $t_2 > t''$ and $t_1 \leq t''$, input port i must send out exactly one packet in UFS mode every time slot throughout $[t'', t_2]$. $\sum_{j,m=1}^{N} C_{ijm}^{U}$ should then be decremented by 1 upon every departure of such packets since none of the UFS evacuation phase bins at input port i has stolen service from t'' to t_2. On the other hand, $\sum_{j,m=1}^{N} C_{ijm}^{U}$ can be incremented by at most 1 every time slot. Therefore $\sum_{j,m=1}^{N} C_{ijm}^{U}$ is strictly not increased throughout $[t'', t_2]$. Hence,

$$\begin{aligned}\sum_{j,m=1}^{N} C_{ijm}^{U}(t_2) &\leq \sum_{j,m=1}^{N} C_{ijm}^{U}(t'') \\ &\leq \sum_{j,m=1}^{N} C_{ijm}^{U}(t') + (t'' - t') \\ &\leq N\end{aligned}$$

- If $t_2 > t''$ and $t_1 > t''$, similarly we can prove that $\sum_{j,m=1}^{N} C_{ijm}^{U}$ is strictly non-increasing throughout $[t_1, t_2]$ and thus

$$\sum_{j,m=1}^{N} C_{ijm}^{U}(t_2) \leq \sum_{j,m=1}^{N} C_{ijm}^{U}(t_1)$$

In summary, we always have

$$\sum_{j,m=1}^{N} C_{ijm}^{U}(t_2) \leq \max\left(N, \sum_{j,m=1}^{N} C_{ijm}^{U}(t_1)\right)$$

□

Now we are ready to prove Lemma 2.

Proof of Lemma 2

Proof. Whenever B_{ijm} is in UFS evacuation phase, C_{ijm}^{U} will be incremented by 1 upon the arrival of every packet and be decremented by *at most* 1 upon the departure of every

packet. Thus we have

$$B_{ijm}(t_2) - B_{ijm}(t_1) \leq C^U_{ijm}(t_2) - C^U_{ijm}(t_1) \leq C^U_{ijm}(t_2)$$

Therefore it's sufficient to prove that $C^U_{ijm}(t_2) \leq N^3$.

If none of the bins at input port i has ever stolen service throughout $[0, t_2]$, by Lemma 7, we have

$$\sum_{j,m=1}^{N} C^U_{ijm}(t_2) \leq \max\left(N, \sum_{j,m=1}^{N} C^U_{ijm}(0)\right) = N \leq N^3$$

Otherwise, let $t' \geq 1$ be the time such that none of the UFS-ready bins at input port i has ever stolen service throughout time slots t' to t_2, but some bin, say $B_{ij'm'}$, steals service in time slot $(t'-1)$ to send out a packet pkt. If so, $B_{ij'm'}$ should have been scheduled to send the first packet of the frame, to which pkt belongs, at some time slot $t'' \in [t'-N,\ t'-1]$. At that time, we must have $C^U_{ij'm'}(t'') \leq N-1$ and $C^U_{ij'm'}(t'') = \max_{j,m=1}^{N} C^U_{ijm}(t'')$. Thus we have

$$\begin{aligned}
\sum_{j,m=1}^{N} C^U_{ijm}(t') &\leq \sum_{j,m=1}^{N} C^U_{ijm}(t'') + (t' - t'') \\
&\leq N^2(N-1) + N \\
&\leq N^3
\end{aligned}$$

Since no bin has stolen services throughout $[t', t_2]$, by Lemma 7, we have

$$C^U_{ijm}(t_2) \leq \sum_{j,m=1}^{N} C^U_{ijm}(t_2) \leq \max\left(N, \sum_{j,m=1}^{N} C^U_{ijm}(t')\right) \leq N^3$$

□

B.9 Relabel $\{B_{ijm}\}_{i,m=1}^{N}$ to $\{B_{(k)}\}_{k=1}^{N^2}$

We relabel the N^2 input port bins $\{B_{ijm}\}_{i,m=1}^{N}$ according to the end time of their last UFS waiting phase before T_2 as follows. Let $[t_1^{[ijm]}, t_2^{[ijm]}]$, $i, m = 1, 2, \ldots, N$, be the last UFS waiting phase of B_{ijm} before T_2. Now we temporarily rewrite the term $t_2^{[ijm]}$ as $t_2^{(i,j,m)}$, and B_{ijm} as $B_{(i,j,m)}$, in order to introduce the following "(reversed) order statistics" of $t_2^{[ijm]}$ and the relabeling accordingly of B_{ijm}, for $i, m = 1, 2, \ldots, N$. Define $\mathcal{V}_j$ as

$$\mathcal{V}_j = \{(i, j, m) \mid i, m = 1, 2, \ldots, N\}$$

Let $\sigma : \mathcal{V}_j \to \{1, 2, \ldots, N^2\}$ be a bijection such that $t_2^{\sigma^{-1}(1)} \geq t_2^{\sigma^{-1}(2)} \geq \cdots \geq t_2^{\sigma^{-1}(N^2)}$. To make our presentation more succinct, we use $B_{(k)}$, $t_1^{(k)}$ and $t_2^{(k)}$ as shorthands for $B_{\sigma^{-1}(k)}$, $t_1^{\sigma^{-1}(k)}$ and $t_2^{\sigma^{-1}(k)}$, respectively. Thus, for any integer $k \in [1, N^2]$, we always have[2] $t_2^{(k)} \leq t_2^{(k-1)} \leq T_2$, and $B_{(k)}$ should never be in UFS waiting phase throughout $[t_2^{(k)}, T_2]$, as shown in Figure 4.3 (a).

B.10 Proof of Theorem 2(b) and 2(c)

In the following we will prove Theorem 4, which implies Theorems 2(b) and 2(c).

Theorem 4.

$$\limsup_{t\to\infty} \frac{G_{jm}(t)}{t} = 0 \quad j, m = 1, 2, \ldots, N$$

Proof. We prove the theorem by contradiction. Suppose there exists a pair of integers $j > 0$ and $m > 0$ such that

$$\limsup_{t\to\infty} \frac{G_{jm}(t)}{t} = \gamma > 0 \tag{B.1}$$

Thus, for $\epsilon = \frac{\gamma}{4}$, there exists an integer $T' > 0$, such that

$$t > T' \Rightarrow \frac{G_{jm}(t)}{t} < (\gamma + \epsilon)$$

[2]For rigorousness, one can define $t_2^{(0)} = T_2$.

By Lemma 3, for $p = \frac{\gamma-\epsilon}{N}$, there exists an integer $T_A > 0$, such that

$$\left\{ \begin{array}{l} T > T_A \\ 0 \le t_1 < t_2 \le T \\ t_2 - t_1 \ge pT \end{array} \right\} \Rightarrow A_j(t_2) - A_j(t_1) \le (1+\epsilon)(t_2 - t_1) \tag{B.2}$$

From (B.1), for the same ϵ, there exists an increasing sequence $\{T_2^k\}_{k=1}^{\infty}$ such that $\lim_{k\to\infty} T_2^k = \infty$, and for $k = 1, 2, \ldots$ we have $T_2^k > T_A$ and $\frac{G_{jm}(T_2^k)}{T_2^k} > (\gamma - \epsilon)$.

As $G_{jm}(0) = 0$ and $G_{jm}(T_2^k) > 0$, for each T_2^k, there must exist a $T_1^k < T_2^k$ such that $G_{jm}(T_1^k) = 0$ and $G_{jm}(t) > 0$ for any $t \in [T_1^k + 1, T_2^k]$.

Note that in the following proof, k (as well as k') is not an exponent in the terms T_1^k and T_2^k, but 3 in the term $5N^3C$ is. By Lemma 5, we have

$$\begin{aligned} & D_j(T_2^k) - D_j(T_1^k) \\ & \ge (T_2^k - T_1^k) + NG_{jm}(T_2^k) - 5N^3C \\ & \ge (T_2^k - T_1^k) + N(\gamma - \epsilon)T_2^k - 5N^3C \end{aligned} \tag{B.3}$$

Furthermore, since $G_{jm}(T_1^k) = 0$, we have

$$N \cdot (T_2^k - T_1^k) \ge I_{jm}(T_2^k) - I_{jm}(T_1^k) \ge G_{jm}(T_2^k) \ge (\gamma - \epsilon)T_2^k$$

Therefore $T_2^k - T_1^k \ge \frac{(\gamma-\epsilon)}{N} T_2^k = pT_2^k$. Since $T_2^k > T_A$, by (B.2), we have

$$A_j(T_2^k) - A_j(T_1^k) \le (1+\epsilon)(T_2^k - T_1^k) \tag{B.4}$$

Combining (B.3) and (B.4), we have

$$\begin{aligned} B_j(T_1^k) &= B_j(T_2^k) + \big(D_j(T_2^k) - D_j(T_1^k)\big) - \big(A_j(T_2^k) - A_j(T_1^k)\big) \\ &\ge \big(D_j(T_2^k) - D_j(T_1^k)\big) - \big(A_j(T_2^k) - A_j(T_1^k)\big) \end{aligned}$$

$$\geq (T_2^k - T_1^k) + N(\gamma - \epsilon)T_2^k - 5N^3C - (1+\epsilon)(T_2^k - T_1^k)$$
$$\geq N(\gamma - \epsilon)T_2^k - 5N^3C - \epsilon T_2^k$$
$$\geq N(\gamma - 2\epsilon)T_2^k - 5N^3C$$

We consider two cases:

(i) If $\max_{k=1}^{\infty}\{T_1^k\} < \infty$, there must exist an integer $k' > 0$ such that $T_2^{k'} > \frac{N\max_{k=1}^{\infty}\{T_1^k\} + 5N^3C}{N(\gamma - 2\epsilon)}$, since $\lim_{k\to\infty} T_2^k = \infty$. This however implies $B_j(T_1^{k'}) \geq N(\gamma - 2\epsilon)T_2^{k'} - 5N^3C > N\max_{k=1}^{\infty}\{T_1^k\} \geq NT_1^{k'}$, which contradicts the fact that $B_j(T_1^{k'}) \leq A_j(T_1^{k'}) \leq NT_1^{k'}$ (due to (4.3)).

(ii) Otherwise, we must have $T_1^k \to \infty$ as $k \to \infty$, thus

$$\begin{aligned}\limsup_{k\to\infty} \frac{B_j(T_1^k)}{T_1^k} &\geq \limsup_{k\to\infty} \frac{N(\gamma - 2\epsilon)T_2^k - 5N^3C}{T_1^k} \\ &\geq \limsup_{k\to\infty} \frac{N(\gamma - 2\epsilon)T_1^k - 5N^3C}{T_1^k} \\ &= N(\gamma - 2\epsilon) \\ &> 0\end{aligned}$$

This contradicts Theorem 3.

□

B.11 Stability proof of the modified SRS with Hawking radiation

In this section, we use fluid model [46] to prove the stability of the modified SRS scheme described in Section 4.4.

B.11.1 Modeling and assumptions

To formulate the fluid model of our system, we need to make a slightly different assumption on the packet arrival process. Recall that $A_{ijm}(t)$ is the cumulative number of

packet arrivals into bin B_{ijm} by time slot t (since time 0). Assume that $\{A_{ijm}(\cdot), i, j, m = 1, 2, \ldots, N\}$ are independent stochastic processes[3] that satisfy a strong law of large numbers (SLLN) with probability one, i.e., there exists constants $\lambda_{ijm} \in [0, 1]$, $i, j, m = 1, \ldots, N$, such that

$$\lim_{t\to\infty} \frac{A_{ijm}(t)}{t} = \lambda_{ijm} \qquad i, j, m = 1, \ldots, N \tag{B.5}$$

For simplicity, here we will assume that the packet arrival process for each input port are *i.i.d.*, and independent across input ports. Note that, since we use fluid limit techniques, this assumption can be relaxed in many different ways at the cost of additional notation. In particular, we only need a Markovian description of the queueing system for our results to hold.

Let $D_{ijm}(t)$ be the cumulative number of packet depart from bin B_{ijm} by time slot t (since time 0). Let $O_{ijm}(t)$ be the cumulative number of packet depart from queue group G_{jm} by time slot t (since time 0). The system can then be modeled as a stochastic process $(B(t), G(t), D(t), O(t))$, where

$$\begin{aligned}
B(t) &\triangleq \left\{ B_{ijm}(t) \,\middle|\, i, j, m = 1, 2, \ldots, N \right\} \\
G(t) &\triangleq \left\{ G_{jm}(t) \,\middle|\, j, m = 1, 2, \ldots, N \right\} \\
D(t) &\triangleq \left\{ D_{ijm}(t) \,\middle|\, i, j, m = 1, 2, \ldots, N \right\} \\
O(t) &\triangleq \left\{ O_{jm}(t) \,\middle|\, j, m = 1, 2, \ldots, N \right\}
\end{aligned}$$

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be the probability space that this stochastic process is defined on, where Ω is the sample space, $\mathcal{F}$ is a σ-field on Ω, and $\mathbb{P}$ is the probability measure on $(\Omega, \mathcal{F})$. We shall

[3]Note that, for simplicity, when proving the (rate) stability of the baseline SRS in Section 4.3, we assume that the packet arrival process is deterministic and satisfies the admissible condition in (4.2). Equivalently, one can also model the packet arrivals as a stochastic process that satisfies the SLLN claim in (B.5), and adapt the derivations in Section 4.3 with some minor changes to prove that the baseline SRS can achieve 100% throughput with probability one (actually, one can easily prove that all the results in Section 4.3 continues to hold for the sample paths that satisfy the SLLN claim in (B.5)).

sometimes use the notations $B(\cdot,\omega)$, $G(\cdot,\omega)$, $D(\cdot,\omega)$ and $O(\cdot,\omega)$ to explicitly denote the dependency on the sample path $\omega \in \Omega$.

We claim that, given the modified SRS scheme described in Section 4.4, the joint process $\{(B(t), G(t), D(t), O(t))\}_{t=0}^{\infty}$, resulting from the modified SRS scheme and any *i.i.d.* arrival process $A(t) \triangleq (A_{ijm}(\cdot), i, j, m = 1, 2, \ldots, N)$, is a Markov chain. This property is clear from the fact that, when working under the modified SRS scheme, $B(t)$, $G(t)$, $D(t)$ and $O(t)$ are functions of only $B(t-1)$, $G(t-1)$, $D(t-1)$, $O(t-1)$, and the random packet arrival vector $a(t) \triangleq A(t) - A(t-1)$ that is independent of all other random vectors. We'll refer to this Markov chain as *the ambient Markov chain* of the system in the sequel.

We claim this Markov chain is irreducible and aperiodic via the following assumption: Given any time slot t, there's a positive probability that no packet arrives in this time slot, i.e., $P[A(t) - A(t-1) = 0] > 0$ for any $t > 0$. Here we provide only a sketchy justification. To justify the irreducibility, we show that $B(t)$, $G(t)$, $D(t)$ and $O(t)$, starting from any state they are currently in, will with a nonzero probability return to the "all-queues-empty" state in a finite number of time slots. To show this property, we claim that, for any integer $\tau > 0$, the switch could, with a nonzero probability, have no packet arrivals to any of its input ports during $[t, t+\tau]$. This claim is true because, the arrival process $A(t)$ is *i.i.d.*, and we have $P[A(t') - A(t'-1) = 0] > 0$ for any $t' \in [t, t+\tau]$. Furthermore, when the switch is nonempty, at least $(T-1)$ packets will be transferred (at least) one step further towards the output ports (from the input port to the intermediate port, or from the intermediate port to the output port) within every T cycles. Hence, when there are no packet arrivals during $[t, t+\tau]$, which happens with a nonzero probability, the modified SRS scheme can clear all the queues during $[t, t+\tau]$, with a sufficiently large τ, and return the Markov chain to the "all-queues-empty" state. Therefore, the Markov chain is irreducible. To justify the aperiodicity of the Markov chain, we note that there is a nonzero probability for the Markov chain to stay at "all-queues-empty" for at least two consecutive time slots.

In addition, to formulate the fluid model, we extend the above discrete time functions

to the continuous time domain. More specifically, for $t \in [0, +\infty)$, we define

$$
\begin{aligned}
&B_{ijm}(t) = B_{ijm}(\lfloor t \rfloor) \qquad i, j, m = 1, 2, \ldots, N \\
&G_{jm}(t) = G_{jm}(\lfloor t \rfloor) \qquad j, m = 1, 2, \ldots, N \\
&D_{ijm}(t) = D_{ijm}(\lfloor t \rfloor) + (t - \lfloor t \rfloor)\big(D_{ijm}(\lceil t \rceil) - D_{ijm}(\lfloor t \rfloor)\big) \qquad i, j, m = 1, 2, \ldots, N \\
&O_{jm}(t) = O_{jm}(\lfloor t \rfloor) + (t - \lfloor t \rfloor)(O_{jm}(\lceil t \rceil) - O_{jm}(\lfloor t \rfloor)) \qquad j, m = 1, 2, \ldots, N
\end{aligned}
$$

where $\lceil t \rceil$ is the largest integer that is smaller than or equal to t and $\lfloor t \rfloor$ is the smallest integer that is larger than or equal to t.

B.11.2 Basic fluid model equations

Let $\mathcal{X}$ be the state space of the Markov chain $(B(t), G(t), D(t), O(t))$. Let (B^x, G^x, D^x, O^x) be the Markov process with initial state $x \in \mathcal{X}$. We define.

$$
|x| \triangleq \sum_{i,j,m=1}^{N} B^x_{ijm}(0) + \sum_{j,m=1}^{N} G^x_{jm}(0)
$$

where $\| \cdot \|_1$ is the 1-norm.

Now, for each sample path $\omega \in \Omega$, we define fluid scaled processes

$$
\Big(\hat{B}^x(t,\omega), \hat{G}^x(t,\omega), \hat{D}^x(t,\omega), \hat{O}^x(t,\omega)\Big) \triangleq \frac{1}{|x|}\Big(\hat{B}^x(|x|t,\omega), \hat{G}^x(|x|t,\omega), \hat{D}^x(|x|t,\omega), \hat{O}^x(|x|t,\omega)\Big)
$$

Proposition 1 (Fluid Model)**.** *Fix a sample path $\omega \in \Omega$ such that the SLLN claim in (B.5) is true. For any unbounded set $C \subset \mathcal{X}$ of initial states, there exists a sequence $\{x_k\} \subset C$ with $|x_k| \to \infty$ and Lipschitz continuous functions $(\hat{B}, \hat{G}, \hat{D}, \hat{O})$, such that*

$$
\Big(\hat{B}^{x_k}(\cdot,\omega), \hat{G}^{x_k}(\cdot,\omega), \hat{D}^{x_k}(\cdot,\omega), \hat{O}^{x_k}(\cdot,\omega)\Big) \to \Big(\hat{B}, \hat{G}, \hat{D}, \hat{O}\Big) \quad u.o.c \quad \textit{as } k \to \infty \tag{B.6}
$$

where the convergence is uniform on compact sets (u.o.c). The four-tuple $(\hat{B}, \hat{G}, \hat{D}, \hat{O})$ *is said to be a fluid limit path of the system. It satisfies the following fluid equations*

$$\hat{B}_{ijm}(t) = \hat{B}_{ijm}(0) + \lambda_{ijm}t - \hat{D}_{ijm}(t) \qquad i, j, m = 1, \ldots, N$$

$$\hat{G}_{jm}(t) = \hat{G}_{jm}(0) + \sum_{i=1}^{N} \hat{D}_{ijm}(t) - \hat{O}_{jm}(t) \qquad j, m = 1, \ldots, N$$

$$\hat{B}_{ijm}(t) \geq 0 \qquad i, j, m = 1, \ldots, N$$

$$\hat{G}_{jm}(t) \geq 0 \qquad j, m = 1, \ldots, N$$

$$\hat{D}_{ijm}(0) = 0, \quad \hat{D}_{ijm}(\cdot) \text{ is non-decreasing} \qquad i, j, m = 1, \ldots, N$$

$$\hat{O}_{jm}(0) = 0, \quad \hat{O}_{jm}(\cdot) \text{ is non-decreasing} \qquad j, m = 1, \ldots, N$$

$$\hat{D}_{ijm}(t) - \hat{D}_{ijm}(s) \leq t - s \text{ for } 0 \leq s < t \qquad i, j, m = 1, \ldots, N$$

$$\hat{O}_{jm}(t) - \hat{O}_{jm}(s) \leq t - s \text{ for } 0 \leq s < t \qquad j, m = 1, \ldots, N$$

A solution for the above equations is called a fluid model solution.

The proof of Proposition 1 is somewhat standard. We refer the reader to [55].

B.11.3 Stability results

We first give the formal definition of stabilities.

Definition 1 (Stability)**.** *The system is said to be stable if its ambient Markov chain is positive recurrent.*

The main result of this section is stated as follows

Theorem 5. *The switch is stable when working under the modified SRS scheme described in Section 4.4, if we have*

$$\sum_{j,m=1}^{N} \lambda_{ijm} < 1 \qquad i = 1, \ldots, N$$

$$\sum_{i,m=1}^{N} \lambda_{ijm} < 1 - \frac{1}{T} \qquad j = 1, \ldots, N$$

We'll give the proof of Theorem 5 using fluid model in the rest of this section. More specifically, we'll first prove that the corresponding fluid model is stable as defined in Definition 2 and the stability of the original system is then guaranteed by Proposition 2.

Definition 2 (Fluid Stability)**.** *The fluid model given in Proposition 1 is said to be stable if there exists a time $t_0 \geq 0$ such that, for each fluid limit path with $\sum_{i,j,m=1}^{N} \hat{B}_{ijm}(0) + \sum_{j,m=1}^{N} \hat{G}_{jm}(0) = 1$, we have $\hat{B}(t) = 0$ and $\hat{G}(t) = 0$ for all $t \geq t_0$.*

Proposition 2. *For a given arrival rate vector $\lambda \triangleq (\lambda_{ijm} : i, j, m = 1, \ldots, N)$, the system is stable if its ambient Markov chain is irreducible and the corresponding fluid model is stable [46].*

B.11.4 Proof of Theorem 5

Because the fluid limit path $(\hat{B}, \hat{G}, \hat{D}, \hat{O})$ is Lipschitz continuous and hence differentiable almost everywhere, the set of non-differentiable points has Lebesgue measure zero and can be excluded in verifying stability of the fluid model [46]. For convenience, we define

Definition 3 (Regular Point)**.** *A point $t > 0$ is said to be a regular point for a fluid limit path $(\hat{B}, \hat{G}, \hat{D}, \hat{O})$ if all of its components are differentiable at t.*

We have the following lemma if the switch works under the modified SRS scheme with Hawking radiation.

Lemma 8. *Assume the switch works under the modified SRS scheme with Hawking radiation. Let t be a regular point of a fluid limit path $(\hat{B}, \hat{G}, \hat{D}, \hat{O})$, we have*

$$\hat{B}_{ijm}(t) > 0 \quad \Longrightarrow \quad \frac{d}{dt}\left(\sum_{j',m'=1}^{N} \hat{D}_{ij'm'}(t)\right) = 1 \qquad i, j, m = 1, \ldots, N \tag{B.7}$$

$$\hat{G}_{jm}(t) > 0 \quad \Longrightarrow \quad \frac{d}{dt}\left(\sum_{m'=1}^{N} \hat{O}_{jm'}(t)\right) \geq 1 - \frac{1}{T} \qquad j, m = 1, \ldots, N \tag{B.8}$$

Proof. (a) *Proof of (B.7).*

Suppose $(\hat{B}, \hat{G}, \hat{D}, \hat{O})$ is a fluid limit path. Fix a sample path $\omega \in \Omega$ such that (B.5) and (B.6) hold. There exists a sequence of initial stats $\{x_k\}$ with $|x_k| \to \infty$ as $k \to \infty$ such that

$$\left(\hat{B}^{x_k}(\cdot,\omega), \hat{G}^{x_k}(\cdot,\omega), \hat{D}^{x_k}(\cdot,\omega), \hat{O}^{x_k}(\cdot,\omega)\right) \to \left(\hat{B}, \hat{G}, \hat{D}, \hat{O}\right) \quad u.o.c \quad \textit{as } k \to \infty$$

Assume $\hat{B}_{ijm}(t) = \epsilon > 0$. By the continuity of $\hat{B}(\cdot)$, there exists $\delta > 0$ such that

$$\hat{B}_{ijm}(s) \geq \frac{\epsilon}{2} \qquad \textit{for } s \in [t-\delta, t+\delta]$$

As $\limsup_{k\to\infty} |\hat{B}^{x_k}_{ijm}(t) - \hat{B}_{ijm}(t)| \to 0$, there exists $K > 0$ such that for any $k > K$ we have

$$\sup |\hat{B}^{x_k}_{ijm}(s,\omega) - \hat{B}_{ijm}(s)| \leq \frac{\epsilon}{4} \qquad \textit{for } s \in [t-\delta, t+\delta]$$

Thus for $k > K$, we have

$$\hat{B}^{x_k}_{ijm}(s,\omega) \geq \frac{\epsilon}{4} \qquad \textit{for } s \in [t-\delta, t+\delta]$$

i.e.,

$$B^{x_k}_{ijm}(s,\omega) \geq \frac{\epsilon}{4}|x_k| \qquad \textit{for } s \in [(t-\delta)|x_k|, (t+\delta)|x_k|]$$

Let $\bar{L}$ be the upper-bound for the length of the H queues, as described in Section 4.4. Since $|x_k| \to \infty$ as $k \to \infty$, there exists $K' > K$ such that

$$|x_k| > \max(\frac{4}{\epsilon}W, \frac{2}{\delta}N\bar{L}T) \qquad \textit{for } k > K' \tag{B.9}$$

where W is the RSP-to-UFS threshold, N is the size of the switch and T is the Hawking radiation period (described in Section 4.4). Then for $k > K'$, we always have $B_{ijm}^{x_k}(s|x_k|, \omega) > W$ for $s \in [(t-\delta)|x_k|, (t+\delta)|x_k|]$, thus B_{ijm} must keep in UFS mode throughout time interval $[(t-\delta)|x_k|, (t+\delta)|x_k|]$. Then one of the following statements must be true

- If B_{ijm} is in UFS-evacuation phase at time $(t-\delta)|x_k|$, it must keep in UFS-evacuation phase throughout time interval $[(t-\delta)|x_k|, (t+\delta)|x_k|]$. Then input port i must send out one packet per time slot in UFS mode throughout time interval $[(t-\delta)|x_k|, (t+\delta)|x_k|]$. Thus we have

$$\sum_{j',m'=1}^{N} D_{ij'm'}^{x_k}(u_2, \omega) - \sum_{j',m'=1}^{N} D_{ij'm'}^{x_k}(u_1, \omega) = u_2 - u_1$$

  for $u_1 \leq u_2$, $u_1, u_2 \in [(t-\delta)|x_k|, (t+\delta)|x_k|] \subset [(t-\frac{\delta}{2})|x_k|, (t+\frac{\delta}{2})|x_k|]$.

- If B_{ijm} is in UFS-accumulation phase at time $(t-\delta)|x_k|$, it has to wait H_{ijm} to be cleaned. But since $H_{ijm}^{x_k}((t-\delta)|x_k|, \omega) \leq \bar{L}$ (as the lengths of the H queues are upper-bounded by $\bar{L}$), it will be cleaned within (at most) $N\bar{L}T$ time slots. Thus B_{ijm} will enter UFS-evacuation phase before time $(t-\delta)|x_k| + N\bar{L}T$ and keeps in UFS-evacuation phase throughout time interval $[(t-\delta)|x_k| + N\bar{L}T, (t+\delta)|x_k|]$. From (B.9), we know that $[(t-\frac{\delta}{2})|x_k|, (t+\frac{\delta}{2})|x_k|] \subset [(t-\delta)|x_k| + N\bar{L}T, (t+\delta)|x_k|]$ as $|x_k| > \frac{2}{\delta}N\bar{L}T$. Therefore, we have

$$\sum_{j',m'=1}^{N} D_{ij'm'}^{x_k}(u_2, \omega) - \sum_{j',m'=1}^{N} D_{ij'm'}^{x_k}(u_1, \omega) = u_2 - u_1$$

  for $u_1 \leq u_2$, $u_1, u_2 \in [(t-\frac{\delta}{2})|x_k|, (t+\frac{\delta}{2})|x_k|]$.

In summary, when $k > K'$, we always have

$$\sum_{j',m'=1}^{N} D_{ij'm'}^{x_k}(u_2, \omega) - \sum_{j',m'=1}^{N} D_{ij'm'}^{x_k}(u_1, \omega) = u_2 - u_1$$

for $u_1 \leq u_2$, $u_1, u_2 \in [(t-\frac{\delta}{2})|x_k|, (t+\frac{\delta}{2})|x_k|]$. Therefore, for any $u_1, u_2 \in [(t-\frac{\delta}{2}), (t+\frac{\delta}{2})]$ with $u_1 \leq u_2$, we have

$$\sum_{j',m'=1}^{N} D_{ij'm'}^{x_k}(u_2|x_k|, \omega) - \sum_{j',m'=1}^{N} D_{ij'm'}^{x_k}(u_1|x_k|, \omega) = u_2|x_k| - u_1|x_k|$$

i.e.,

$$\sum_{j',m'=1}^{N} \hat{D}_{ij'm'}^{x_k}(u_2, \omega) - \sum_{j',m'=1}^{N} \hat{D}_{ij'm'}^{x_k}(u_1, \omega) = u_2 - u_1$$

Taking the limit as $k \to \infty$, we have

$$\sum_{j',m'=1}^{N} \hat{D}_{ij'm'}(u_2) - \sum_{j',m'=1}^{N} \hat{D}_{ij'm'}(u_1) = u_2 - u_1$$

for any $u_1, u_2 \in [(t-\frac{\delta}{2}), (t+\frac{\delta}{2})]$ with $u_1 \leq u_2$, from which (B.7) follows.

(b) *Proof of (B.8).*

Assume $\hat{G}_{jm}(t) = \epsilon > 0$. Similarly, for the same fluid limit path $(\hat{B}, \hat{G}, \hat{D}, \hat{O})$, sample path ω and sequence $\{x_k\}$, we can prove that there exists $K > 0$ such that for $k > K$, we have

$$G_{jm}^{x_k}(s, \omega) \geq \frac{\epsilon}{4}|x_k| \qquad \textit{for } s \in [(t-\delta)|x_k|, (t+\delta)|x_k|]$$

Since $|x_k| \to \infty$ as $k \to \infty$, there exists $K' > K$ such that $|x_k| > \frac{4}{\epsilon} N\bar{L}$ for $k > K'$. Then for any $k > K'$, we always have $G_{jm}^{x_k}(s, \omega) > N\bar{L}$ for $s \in [(t-\delta)|x_k|, (t+\delta)|x_k|]$. Thus, with a slight abuse of notation, we have

$$\begin{aligned} U_{jm}^{x_k}(s, \omega) &= G_{jm}^{x_k}(s, \omega) - \sum_{i'=1}^{N} H_{i'jm}^{x_k}(s, \omega) \\ &\geq G_{jm}^{x_k}(s, \omega) - N\bar{L} \\ &> 0 \end{aligned}$$

for $s \in [(t-\delta)|x_k|, (t+\delta)|x_k|]$, which implies that all the N intermediate port queues $U_{jm'}$, $m' = 1, \ldots, N$, are non-empty throughout time interval $[(t-\delta)|x_k|, (t+\delta)|x_k|]$. Thus we have

$$\begin{aligned}\sum_{m'=1}^{N} O_{jm'}^{x_k}(u_2, \omega) - \sum_{m'=1}^{N} O_{jm'}^{x_k}(u_1, \omega) &\geq \left\lfloor (u_2 - u_1) \cdot (1 - \frac{1}{T}) \right\rfloor \\ &\geq (u_2 - u_1)(1 - \frac{1}{T}) - 1\end{aligned}$$

for $u_1 \leq u_2$, $u_1, u_2 \in [(t-\delta)|x_k|, (t+\delta)|x_k|]$. Therefore, for any $u_1, u_2 \in [(t-\delta), (t+\delta)]$ with $u_1 \leq u_2$, we have

$$\sum_{m'=1}^{N} O_{jm'}^{x_k}(u_2|x_k|, \omega) - \sum_{m'=1}^{N} O_{jm'}^{x_k}(u_1|x_k|, \omega) \geq (u_2|x_k| - u_1|x_k|)(1 - \frac{1}{T}) - 1$$

i.e.,

$$\sum_{m'=1}^{N} \hat{O}_{jm'}^{x_k}(u_2, \omega) - \sum_{m'=1}^{N} \hat{O}_{jm'}^{x_k}(u_1, \omega) \geq (u_2 - u_1)(1 - \frac{1}{T}) - \frac{1}{|x_k|}$$

Taking the limit as $k \to \infty$, we have

$$\sum_{m'=1}^{N} \hat{O}_{jm'}(u_2) - \sum_{m'=1}^{N} \hat{O}_{jm'}(u_1) \geq (u_2 - u_1)(1 - \frac{1}{T}) \tag{B.10}$$

for any $u_1, u_2 \in [(t-\delta), (t+\delta)]$ with $u_1 \leq u_2$. (B.8) then follows from (B.10) as t is a regular point. □

Theorem 5 is implied by Proposition 2 and the following theorem.

Theorem 6. *Assume the system works under the modified SRS scheme with Hawking radiation. Then the fluid model is stable if we have*

$$\begin{aligned}\sum_{j,m=1}^{N} \lambda_{ijm} &< 1 \qquad\qquad i = 1, \ldots, N \\ \sum_{i,m=1}^{N} \lambda_{ijm} &< 1 - \frac{1}{T} \qquad j = 1, \ldots, N\end{aligned}$$

Proof. Define a Lyapunov function $f(t)$ as

$$f(t) = \max(f_1(t), f_2(t))$$

where

$$f_1(t) = \max_{i=1,\ldots,N} \left(N^2 \sum_{j,m=1}^{N} \hat{B}_{ijm}(t) \right)$$
$$f_2(t) = \max_{j=1,\ldots,N} \left(\sum_{m=1}^{N} \hat{G}_{jm}(t) + \sum_{i,m=1}^{N} \hat{B}_{ijm}(t) \right)$$

It's easy to prove that $f(t) \geq 0$ and $f(t) = 0 \Leftrightarrow \sum_{j,m=1}^{N} \hat{G}_{jm}(t) + \sum_{i,j,m=1}^{N} \hat{B}_{ijm}(t) = 0$. Let t be a regular point of the fluid model. When $f(t) > 0$, one of the following statements must be true.

(1) If $f_1(t) \geq f_2(t)$, we must have $f(t) = f_1(t) = N^2 \sum_{j,m=1}^{N} \hat{B}_{ijm}(t)$ for some input port i. Then, from Proposition 1, we have

$$\begin{aligned} f(t) &= N^2 \sum_{j,m=1}^{N} \hat{B}_{ijm}(t) \\ &= N^2 \sum_{j,m=1}^{N} \left(\hat{B}_{ijm}(0) + \lambda_{ijm} t - \hat{D}_{ijm}(t) \right) \\ &= N^2 \sum_{j,m=1}^{N} \hat{B}_{ijm}(0) + N^2 \sum_{j,m=1}^{N} \lambda_{ijm} t - N^2 \sum_{j,m=1}^{N} \hat{D}_{ijm}(t) \end{aligned}$$

Since $f(t) > 0$, we have $\sum_{j,m=1}^{N} \hat{B}_{ijm}(t) > 0$. According to Lemma 8, we know that

$$\frac{d}{dt} \left(\sum_{j,m=1}^{N} \hat{D}_{ijm}(t) \right) = 1$$

Thus

$$\begin{aligned}\frac{d}{dt}f(t) &= N^2 \sum_{j,m=1}^{N} \lambda_{ijm} - N^2 \left(\sum_{j,m=1}^{N} \hat{D}_{ijm}(t) \right) \\ &= N^2 \left(\sum_{j,m=1}^{N} \lambda_{ijm} - 1 \right) \\ &< 0 \end{aligned} \tag{B.11}$$

(2) If $f_1(t) < f_2(t)$, we must have $f(t) = f_2(t) = \sum_{m=1}^{N} \hat{G}_{jm}(t) + \sum_{i,m=1}^{N} \hat{B}_{ijm}(t)$ for some output port j.

Note that we must have $\sum_{m=1}^{N} \hat{G}_{jm}(t) > 0$ in such case. Otherwise, suppose $\sum_{m=1}^{N} \hat{G}_{jm}(t) = 0$. Let $\hat{B}_{i'j'm'}(t) = \max_{i,j,m=1,\ldots,N} \hat{B}_{ijm}(t)$, we have

$$\begin{aligned} f_2(t) &= f(t) \\ &= \sum_{i,m=1}^{N} \hat{B}_{ijm}(t) \\ &\leq N^2 \hat{B}_{i'j'm'}(t) \\ &\leq N^2 \sum_{j,m=1}^{N} \hat{B}_{i'jm}(t) \\ &\leq f_1(t) \end{aligned}$$

Contradicted with our assumption that $f_1(t) < f_2(t)$.

Then, from Proposition 1, we have

$$\begin{aligned} f(t) &= \sum_{m=1}^{N} \hat{G}_{jm}(t) + \sum_{i,m=1}^{N} \hat{B}_{ijm}(t) \\ &= \sum_{m=1}^{N} \left(\hat{G}_{jm}(0) + \sum_{i=1}^{N} \hat{D}_{ijm}(t) - \hat{O}_{jm}(t) \right) + \sum_{i,m=1}^{N} \left(\hat{B}_{ijm}(0) + \lambda_{ijm} t - \hat{D}_{ijm}(t) \right) \end{aligned}$$

$$= \sum_{m=1}^{N} \hat{G}_{jm}(0) + \sum_{i,m=1}^{N} \hat{B}_{ijm}(0) + \sum_{i,m=1}^{N} \lambda_{ijm} t - \sum_{m=1}^{N} \hat{O}_{jm}(t)$$

Since $\sum_{m=1}^{N} \hat{G}_{jm}(t) > 0$, from Lemma 8, we know that

$$\frac{d}{dt}\left(\sum_{m=1}^{N} \hat{O}_{jm}(t)\right) \geq 1 - \frac{1}{T}$$

Thus

$$\begin{aligned} \frac{d}{dt} f(t) &= \sum_{i,m=1}^{N} \lambda_{ijm} - \frac{d}{dt}\left(\sum_{m=1}^{N} \hat{O}_{jm}(t)\right) \\ &\leq \sum_{i,m=1}^{N} \lambda_{ijm} - (1 - \frac{1}{T}) \qquad \text{(B.12)} \\ &< 0 \end{aligned}$$

In summary, let $\epsilon \triangleq \min\left(\min_{i=1,\ldots,N}\left(1 - \sum_{j,m=1}^{N} \lambda_{ijm}\right), \min_{j=1,\ldots,N}\left((1 - \frac{1}{T}) - \sum_{i,m=1}^{N} \lambda_{ijm}\right)\right)$. From (B.11) and (B.12), we must have $\frac{d}{dt} f(t) < -\epsilon$ whenever $f(t) > 0$ and t is a regular point. Since $f(0) = 1$ if $\sum_{i,j,m=1}^{N} \hat{B}_{ijm}(0) + \sum_{j,m=1}^{N} \hat{G}_{jm}(0) = 1$, it follows that there exists $t_0 \geq 0$ such that $f(t) = 0$ for all $t \geq t_0$ (see, for example, the proof of Lemma 2 of [56]), proving the theorem. □

APPENDIX C

A THEORETICAL FRAMEWORK ON ANALYZING THE STABILITY OF A BROAD CLASS OF CROSSBAR SCHEDULING ALGORITHMS

In this chapter, we propose a theoretical framework based on the fluid model analysis [46, 57, 58, 59], which can be used to analyze the stability of a broad class (will be defined in Section C.3) of crossbar scheduling algorithms. Note that some definitions and propositions in this Chapter (e.g., Definitions 4 and 6) is identical or similar to those in Appendix B.11.3. We re-state them here for the self-completeness of this chapter.

C.1 Background on the single stage crossbar switches

Most present day switching systems, in Internet routers and data-center switches, employ a single crossbar to interconnect input ports with output ports. A generic input-queued switch is shown in Figure C.1, with N input and N output ports interconnected by a crossbar. Each input port has N Virtual Output Queues (VOQs). A VOQ j at input port i serves as a buffer for packets going from input port i to output port j. The use of VOQs solves the Head-of-Line (HOL) blocking issue [60], which severely limits the throughput of the switch system.

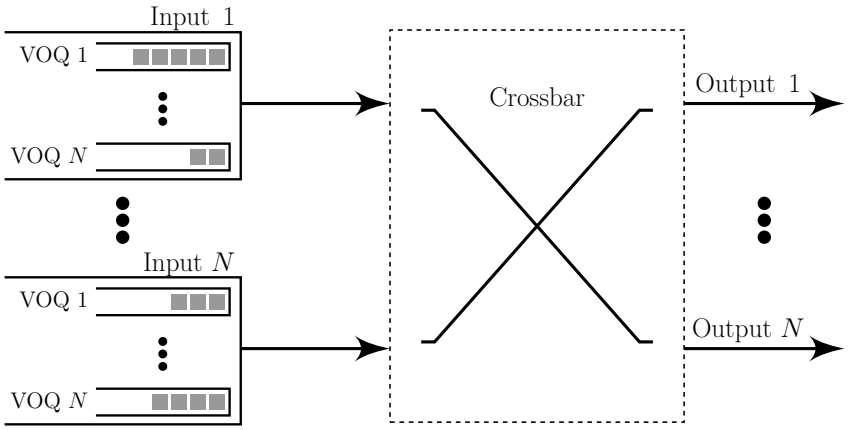


Figure C.1: Generic input-queued crossbar switch.

In an input-queued switch, each input port can be connected to only one output port, and vice versa, in each switching cycle, or time slot. Hence, input-queued switches need to compute, per time slot, a one-to-one *matching* between input and output ports. With the relentless growth in the volume of network traffic across the Internet and in data-centers, switches capable of connecting a large number of ports and operating at very high port/link speeds are badly needed. The primary research challenge when designing such large single-crossbar switch architectures is to develop algorithms that can compute "high quality" matchings – i.e., those that result in high switch throughput (ideally 100%) and low queueing delays for packets – at high speeds.

Unfortunately, there appears to be a tradeoff between the quality of a matching and the time needed to compute it (i.e., computational complexity). Maximum Weight Matching (MWM), with a suitable weight measure, is known to produce (empirically) optimal matchings in terms of queueing delay for a large variety of traffic patterns [61]. Each matching decision however takes $O(N^3)$ time to compute [62]. Researchers have been searching for alternatives [53, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72] that have complexity much lower than $O(N^3)$, but have performance close enough to MWM, as measured by certain performance metrics, in each time slot.

One of the most important performance metrics for a crossbar scheduling algorithm is its stability, or in other words, whether it can achieve 100% throughput for all admissible traffics. In this chapter, we propose a theoretical framework based on the fluid model analysis [46, 57, 58, 59], which can be used to analyze the stability of a broad class of crossbar scheduling algorithms (will be defined shortly).

C.2 Modeling and assumptions

In an input-queued switch, packets arriving at an input port are queued first in their respective VOQs before being switched to their respective output ports by the crossbar. In this chapter, we adopt the standard assumption that all incoming variable-size packets are seg-

mented into fixed-size packets (sometimes referred to as cells), which are then reassembled when leaving the switch. Hence we consider the switching of only fixed-size packets in the sequel, and each such fixed-size packet takes exactly one time slot to transmit. We also make the following standard homogeneity assumption that every input or output link/port has the same maximum transmission rate (normalized to 1), which is equal to that of a transmission line or crosspoint in the crossbar (also normalized to 1).

An $N \times N$ crossbar is generally modeled as a weighted complete bipartite graph, with the N input ports and the N output ports represented as the two disjoint vertex sets respectively. An edge between an input port i and an output port j corresponds to the VOQ j at the input port i, and its weight is the queue length (i.e., the number of packets buffered) of the VOQ. A valid schedule, or *matching*, is a set of edges between the N input ports and the N output ports, in which no two distinct edges share a vertex. Since there can be at most N edges in any such matching, the crossbar can switch at most N packets to their respective output ports during each time slot. Each matching can also be represented as an $N \times N$ sub-permutation matrix[1] $S = \big(s_{ij}\big)$, in which $s_{ij} = 1$ if and only if the input port i is matched with the output port j.

Let $A_{ij}(t)$ be the cumulative number of packet arrivals at the j^{th} VOQ at the input port i by time slot t (since time 0), $i, j = 1, \ldots, N$. Let $D_{ij}(t)$ be the cumulative number of packet departure from the j^{th} VOQ at the input port i by time slot t (since time 0), $i, j = 1, \ldots, N$. Let $Z_{ij}(t)$ be the total number of packets in the j^{th} VOQ at the input port i at time slot t, $i, j = 1, \ldots, N$. Let Π be the set of all $N!$ valid matchings (i.e., the $N \times N$ sub-permutation matrices) from input ports to output ports. Let $\pi(t) \in \Pi$ be the matching matrix selected by the matching algorithm at time slot t, and, with a slight abuse of notation, let $T_\pi(t)$ be the cumulative amount of time that a matching $\pi \in \Pi$ was employed by time slot t.

In the rest of this chapter we flatten all the $N \times N$ matrices defined above, e.g., π and Z, into N^2-dimensional vectors in the row-major order, i.e., the first row of the matrix

[1] An $N \times N$ sub-permutation matrix is an $N \times N$ 0-1 matrix where at most one element in each row or column can take value 1.

becomes the first N scalars in the vector, the second row becomes the next N scalars, and so on. Now that π and Z are vectors, we can take their inner products, denoted as $\langle \cdot, \cdot \rangle$, in the following derivations. For example,$\langle \pi(t), Z(t) \rangle$ is the weight of the schedule (matching) $\pi(t)$, *w.r.t.* the queue length vector $Z(t)$, at time slot t.

Assume the arrival process $A(t) \triangleq \{A_{ij}(\cdot), i, j = 1, 2, \ldots, N\}$ satisfies a strong law of large numbers (SLLN) with probability one, i.e., there exists constants $\lambda_{ij} \geq 0$, $i, j = 1, \ldots, N$, such that

$$\lim_{t \to \infty} \frac{A_{ij}(t)}{t} = \lambda_{ij} \qquad i, j = 1, \ldots, N \tag{C.1}$$

The system can then be modeled as a stochastic process $(Z(t), D(t), T(t))$, where

$$\begin{aligned} Z(t) &\triangleq \left\{ Z_{ij}(t) \,\middle|\, i, j = 1, 2, \ldots, N \right\} \\ D(t) &\triangleq \left\{ D_{ij}(t) \,\middle|\, i, j = 1, 2, \ldots, N \right\} \\ T(t) &\triangleq \left\{ T_{\pi}(t) \,\middle|\, \pi \in \Pi \right\} \end{aligned}$$

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be the probability space that this stochastic process is defined on, where Ω is the sample space, $\mathcal{F}$ is a σ-field on Ω, and $\mathbb{P}$ is the probability measure on $(\Omega, \mathcal{F})$. We shall sometimes use the notations $Z(\cdot, \omega)$, $D(\cdot, \omega)$ and $T(\cdot, \omega)$ to explicitly denote the dependency on the sample path $\omega \in \Omega$.

In addition, to formulate the fluid model, we extend the above discrete time functions to the continuous time domain. Specifically, for $t \in [0, +\infty)$, we define

$$\begin{aligned} A_{ij}(t) &= A_{ij}(\lfloor t \rfloor) \qquad i, j = 1, \ldots, N \\ Z_{ij}(t) &= Z_{ij}(\lfloor t \rfloor) \qquad i, j = 1, \ldots, N \\ D_{ij}(t) &= D_{ij}(\lfloor t \rfloor) + (t - \lfloor t \rfloor)(D_{ij}(\lceil t \rceil) - D_{ij}(\lfloor t \rfloor)) \qquad i, j = 1, \ldots, N \\ T_{\pi}(t) &= T_{\pi}(\lfloor t \rfloor) + (t - \lfloor t \rfloor)(T_{\pi}(\lceil t \rceil) - T_{\pi}(\lfloor t \rfloor)) \qquad \pi \in \Pi \end{aligned}$$

where $\lceil t \rceil$ is the largest integer that is smaller than or equal to t and $\lfloor t \rfloor$ is the smallest integer that is larger than or equal to t.

C.3 The family of (ϵ, δ)-*MWM* switching algorithms with sublinear degradation

Let $W(t) \triangleq \langle \pi(t), Z(t) \rangle$ be the weight of the matching selected by the scheduling algorithm at time slot t. Let $W_{max}(t) \triangleq \max_{\pi} \langle \pi, Z(t) \rangle$ be the weight of the MWM at time slot t. In this chapter, we consider a broad class of switching algorithms that satisfy the following three conditions.

(i) (ϵ, δ)-*MWM Property [73].* A switching algorithm is called (ϵ, δ)-*MWM*, if $\forall \epsilon > 0$, there exists a constant $0 < \delta \leq 1$, such that

$$\mathbb{P}\big[W(t) \geq (1-\epsilon) W_{max}(t)\big] \geq \delta$$

where δ is a constant independent of the time slot t, the queue length vector $Z(t)$ and the scheduling history. Note this δ can depend on ϵ and other (constant) system parameters such as N.

(ii) *Sublinear Degradation.*[2] For $\forall p \in (0, 1)$, there exists a constant $W > 0$, such that we always have

$$\langle \pi(t), Z(t) \rangle \geq (1-p) \cdot \langle \pi(t-1), Z(t) \rangle$$

if $\langle \pi(t-1), Z(t) \rangle > W$, where W is a constant independent of the time slot t, the queue length vector $Z(t)$ and the scheduling history. Note this W can depend on p and other (constant) system parameters such as N.

(iii) *Markov Property.* The matching selected by the switching algorithm at time slot t, i.e., $\pi(t)$, is a random vector that only dependents on $\pi(t-1)$, $Z(t-1)$ and

[2]Note that a function $f(t)$ is called sublinear if and only if, for any given $c > 0$, there exists a t_0 such that $t > t_0 \implies 0 \leq f(t) < c \cdot t$. We name this property as sublinear degradation because the degradation in the "quality" of the matching, i.e., $\langle \pi(t-1), Z(t) \rangle - \langle \pi(t), Z(t) \rangle$, is a sublinear function of $\langle \pi(t-1), Z(t) \rangle$.

$Z(t)$. Similar to that in Appendix B.11.1, it's easy to prove that, given any switch algorithm that satisfies this condition, the joint queueing and scheduling process $\left\{(Z(t), D(t), T(t))\right\}_{t=0}^{\infty}$, under any *i.i.d.* arrival process $A(t)$, is a Markov chain. This Markov chain is irreducible and aperiodic if we further assume that, given any time slot t, there's a positive probability that no packet arrives in this time slot, i.e., $P[A(t) - A(t-1) = 0] > 0$ for any $t > 0$. We'll refer to this Markov chain as *the ambient Markov chain* of the system in the sequel. Note that, since we use fluid limit techniques, this condition can be relaxed in many different ways at the cost of additional notation. In particular, we only need a Markovian description of the queueing system for our results to hold.

C.4 Basic fluid model equations

Let $\mathcal{X}$ be the state space of the Markov chain $(Z(t), D(t), T(t))$. Let (Z^x, D^x, T^x) be the Markov process with initial state $x \in \mathcal{X}$. For each $x \in \mathcal{X}$, we define

$$|x| = \sum_{i,j=1}^{N} Z_{ij}^x(0)$$

Now, for each sample path $\omega \in \Omega$, we define fluid scaled processes

$$\left(\hat{Z}^x(t,\omega), \hat{D}^x(t,\omega), \hat{T}^x(t,\omega)\right) \triangleq \frac{1}{|x|}\left(\hat{Z}^x(|x|t,\omega), \hat{D}^x(|x|t,\omega), \hat{T}^x(|x|t,\omega)\right)$$

Proposition 3 (Fluid Model)**.** *Fix a sample path $\omega \in \Omega$ such that (C.1) is true. For any unbounded set $C \subset \mathcal{X}$ of initial states, there exist a sequence $\{x_k\} \subset C$ with $|x_k| \to \infty$ and Lipschitz continuous functions $(\hat{Z}, \hat{D}, \hat{T})$ such that*

$$\left(\hat{Z}^{x_k}(\cdot,\omega), \hat{D}^{x_k}(\cdot,\omega), \hat{T}^{x_k}(\cdot,\omega)\right) \to \left(\hat{Z}, \hat{D}, \hat{T}\right) \quad u.o.c \quad as\ k \to \infty \tag{C.2}$$

where the convergence is uniform on compact sets (u.o.c). The three-tuple $(\hat{Z}, \hat{D}, \hat{T})$ is

said to be a fluid limit path of the system. It satisfies the following fluid equations

$$\hat{Z}_{ij}(t) = \hat{Z}_{ij}(0) + \lambda_{ij}t - \hat{D}_{ij}(t) \qquad i,j = 1,\dots,N$$

$$\frac{d}{dt}\hat{D}_{ij}(t) = \sum_{\pi\in\Pi}\pi_{ij}\frac{d}{dt}\hat{T}_{\pi}(t),\ \textit{if}\ \hat{Z}_{ij}(t) > 0 \qquad i,j = 1,\dots,N \tag{C.3}$$

$$\sum_{\pi\in\Pi}\hat{T}_{\pi}(t) = t \tag{C.4}$$

$$\hat{Z}_{ij}(t) \geq 0 \qquad i,j = 1,\dots,N$$

$$\hat{T}_{\pi}(0) = 0, \quad \hat{T}_{\pi}(\cdot)\ \textit{is non-decreasing} \qquad \pi \in \Pi$$

$$\hat{T}_{\pi}(t) - \hat{T}_{\pi}(s) \leq t - s\ \textit{for}\ 0 \leq s < t \qquad \pi \in \Pi$$

The proof of Proposition 3 is somewhat standard. We refer the reader to [45] and [55].

C.5 Main results

We first give the formal definition of stabilities.

Definition 4 (Stability)**.** *The switching system is said to be stable if its ambient Markov chain is positive recurrent.*

The main result of this chapter is stated as follows

Theorem 7. *The switch is stable when working under the family of crossbar switching algorithms described in Section C.3, if we have*

$$\sum_{j=1}^{N}\lambda_{ij} < 1 \qquad i = 1,\dots,N$$

$$\sum_{i=1}^{N}\lambda_{ij} < 1 \qquad j = 1,\dots,N$$

We'll give the formal proof of Theorem 7 using fluid model in the next section. More specifically, we'll first prove that the corresponding fluid model is stable as defined in Definition 5 and the stability of the original system is then guaranteed by Proposition 4.

Definition 5 (Fluid Stability)**.** *The fluid model given in Proposition 3 is said to be stable if there exists a time $t_0 \geq 0$ such that, for each fluid limit path with $\sum_{i,j=1}^{N} \hat{Z}_{ij}(0) = 1$, we have $\hat{Z}(t) = 0$ for all $t \geq t_0$.*

Proposition 4. *For a given arrival rate vector $\lambda \triangleq (\lambda_{ij} : i, j = 1, \ldots, N)$, the switching system is stable if its ambient Markov chain is irreducible and the corresponding fluid model is stable [46].*

C.6 Proof of Theorem 7

We first present a pure mathematical lemma (i.e., has nothing to do with switching or networking by itself) that will be used in the stability proof.

Lemma 9. *Let $f : [0, +\infty) \to [0, +\infty)$ be an Lipschitz continuous function. If $\exists\ \delta > 0$ such that $\dot{f}(t) \leq -2\delta\sqrt{f(t)}$ for almost all $t \in [0, +\infty)$ such that f is differentiable at t, we must have $f(t) = 0$ for $t \geq \sqrt{f(0)}/\delta$.*

Proof. Define $g : [0, +\infty) \to [0, +\infty)$ as

$$g(t) = \begin{cases} \left(\sqrt{f(0)} - \delta t\right)^2 & t < \sqrt{f(0)}/\delta \\ 0 & t \geq \sqrt{f(0)}/\delta \end{cases}$$

It's sufficient to prove that $f(t) \leq g(t)$ for $t \geq 0$. Suppose on the contrary that there exists $s_1 \in [0, +\infty)$ such that $f(s_1) > g(s_1)$. Since $g(0) = f(0)$ and both f and g are continuous, by the Intermediate Value Theorem there must exist $s_0 \in [0, s_1)$ such that $f(s_0) = g(s_0)$ and $f(t) \geq g(t)$ for $t \in [s_0, s_1]$. Since both f and g are Lipschitz continuous and $\dot{g}(t) = -2\delta\sqrt{g(t)}$, we have

$$\begin{aligned} f(s_1) - f(s_0) &= \int_{s_0}^{s_1} \dot{f}(t)dt \\ &\leq \int_{s_0}^{s_1} -2\delta\sqrt{f(t)}dt \end{aligned}$$

$$
\begin{aligned}
&\leq \int_{s_0}^{s_1} -2\delta\sqrt{g(t)}dt \\
&= \int_{s_0}^{s_1} \dot{g}(t)dt \\
&= g(s_1) - g(s_0)
\end{aligned}
$$

which implies

$$f(s_1) \leq g(s_1)$$

contradicted with our assumption that $f(s_1) > g(s_1)$.

□

Now, for a given constant $\epsilon \in [0, 1)$ and time slot t, we define $\Delta(t, \epsilon)$ as, starting from time slot t, the number of time slots until the first time that the scheduling algorithm selects a matching whose weight is at least $(1 - \epsilon)$ times the weight of the MWM (i.e., the "upgrading time" from $\pi(t)$ to a matching that is "comparable" to the MWM). Formally, we have

$$\Delta(t, \epsilon) \triangleq \inf\{k \geq 0 \mid W(t + k) \geq (1 - \epsilon)W_{max}(t + k)\}$$

As will be proved in the following lemma, the upgrading time $\Delta(t, \epsilon)$ is a sublinear function of t, if the switch works under a scheduling algorithm that satisfies the (ϵ, δ)-*MWM* condition.

Lemma 10 (Sublinear Upgradating Time)**.** *Given $\epsilon > 0$, for any crossbar switching algorithm that satisfies the (ϵ, δ)-MWM condition, we must have*

$$\lim_{t\to\infty} \frac{\Delta(t, \epsilon)}{t} = 0 \qquad a.s. \tag{C.5}$$

where "a.s." means "almost surely".

Proof. By definition, $\Delta(t,\epsilon)$ is the number of time slots until the first time that the scheduling algorithm selects a matching whose weight is at least $(1-\epsilon)$ times the weight of the MWM. By the (ϵ,δ)-***MWM*** property, there exists $\delta \in (0,1]$, such that $\mathbb{P}\big[W(t) \geq (1-\epsilon)W_{max}(t)\big] \geq \delta$, where δ is a constant independent of the time slot t, the queue length vector $Z(t)$ and the scheduling history. Hence, $\Delta(t,\epsilon)$ can be taken as the number of trials to get the first success, each of these trials has a success probability of at least δ independent of the history. It's then naturally dominated by a geometric distribution random variable with success probability δ. More specifically, for each time slot t, define $\{R_t(k)\}_{k=0}^{\infty}$ as a sequence of random variables such that

$$R_t(k) = \begin{cases} \textit{Uniform}\,[0, p_t(k)] & \textit{if } W(t+k) \geq (1-\epsilon)W_{max}(t+k) \\ \textit{Uniform}\,[p_t(k), 1] & \textit{otherwise} \end{cases}$$

where $p_t(k) \triangleq \mathbb{P}\big[W(t+k) \geq (1-\epsilon)W_{max}(t+k)\big] \geq \delta$, and $\textit{Uniform}\,[a,b]$ represents a random variable that is uniformly distributed in the interval $[a,b]$ and is independent with all the other constants and random variables. Define $Y_t \triangleq \inf\{k \geq 0 \mid R_t(k) \geq \delta\}$. It's then easy to prove that (1) $\{R_t(k)\}_{k=0}^{\infty}$ is a sequence of *i.i.d.* uniformly distribution random variables in the range $[0,1]$; (2) $\{Y_t\}_{t=0}^{\infty}$ is a sequence of *i.i.d.* geometric distribution random variables with success probability δ; (3) for any sample path $\omega \in \Omega$ we always have $Y_t(\omega) \geq \Delta(t,\epsilon,\omega)$. Thus, it's sufficient to prove that

$$\lim_{t\to\infty} \frac{Y_t}{t} = 0 \qquad a.s. \tag{C.6}$$

Since $\mathbf{E}[Y_t^2] = \frac{2-\delta}{\delta^2} < \infty$, by the law of large numbers, with probability one we have

$$\lim_{t\to\infty} \frac{Y_t^2}{t} \leq \lim_{t\to\infty} \frac{1}{t}\sum_{k=1}^{t} Y_k^2 = \frac{2-\delta}{\delta^2} < \infty$$

Then

$$\lim_{t\to\infty}\frac{Y_t^2}{t^2}=\lim_{t\to\infty}\left(\frac{Y_t}{t}\right)^2=0 \qquad a.s.$$

from which (C.6) follows, proving the lemma. □

Because the fluid limit path $(\hat{Z},\hat{D},\hat{T})$ is Lipschitz continuous and hence differentiable almost everywhere, the set of non-differentiable points has Lebesgue measure zero and can be excluded in verifying stability of the fluid model [46]. For convenience, we define

Definition 6 (Regular Point)**.** *A point $t>0$ is said to be a regular point for a fluid limit path $(\hat{Z},\hat{D},\hat{T})$ if all of its components are differentiable at t.*

We have the following lemma if the switch works under a scheduling algorithm that satisfies both the (ϵ,δ)-*MWM* condition and the sublinear degradation condition.

Lemma 11. *Assume the switch works under a scheduling algorithm that satisfies both the (ϵ,δ)-MWM condition and the sublinear degradation condition. Let t be a regular point of a fluid limit path $(\hat{Z},\hat{D},\hat{T})$, we have*

$$\frac{d}{dt}\hat{T}_\pi(t)=0 \text{ if } \langle\pi,\hat{Z}(t)\rangle<\langle\pi',\hat{Z}(t)\rangle \text{ for some } \pi'\in\Pi$$

Proof. Define $\Pi'(t)\triangleq\{\pi'\in\Pi\mid\langle\pi',\hat{Z}(t)\rangle=\max_\pi\langle\pi,\hat{Z}(t)\rangle\}$. It's sufficient to prove that

$$\frac{d}{dt}\hat{T}_\pi(t)=0 \text{ if } \pi\notin\Pi'(t) \tag{C.7}$$

Suppose $(\hat{Z},\hat{D},\hat{T})$ is a fluid limit path. Fix a sample path $\omega\in\Omega$ such that (C.1) and (C.2) hold. There exists a sequence of initial stats $\{x_k\}$ with $|x_k|\to\infty$ as $k\to\infty$ such that

$$\left(\hat{Z}^{x_k}(\cdot,\omega),\hat{D}^{x_k}(\cdot,\omega),\hat{T}^{x_k}(\cdot,\omega)\right)\to\left(\hat{Z},\hat{D},\hat{T}\right) \quad u.o.c \quad as\ k\to\infty \tag{C.8}$$

If $\Pi'(t) \neq \emptyset$, there exist constants $\epsilon_1 \in (0,1)$ and $\epsilon_2 > 0$ such that for any $\pi \notin \Pi'(t)$ and $\pi' \in \Pi'(t)$ we have $(1-\epsilon_1)\langle \pi', \hat{Z}(t)\rangle - \langle \pi, \hat{Z}(t)\rangle \geq \epsilon_2$. By the continuity of $\hat{Z}(\cdot)$, there exists $\tau > 0$ such that

$$(1-\epsilon_1)\langle \pi', \hat{Z}(s)\rangle - \langle \pi, \hat{Z}(s)\rangle \geq \frac{\epsilon_2}{2} \qquad \textit{for } s \in [t-\tau, t+\tau]$$

From the sublinear degradation condition, there exists a constant $W > 0$, such that for any $k > 0$ and time slot $s > 0$ we always have

$$\langle \pi(s), Z^{x_k}(s)\rangle \geq (1-\epsilon_2) \cdot \langle \pi(s-1), Z^{x_k}(s)\rangle \quad \textit{if } \langle \pi(s-1), Z^{x_k}(s)\rangle > W \tag{C.9}$$

By (C.8), there exists $K > 0$, such that for any $k > K$ and $s \in [t-\tau, t+\tau]$, we have $\frac{\epsilon_2}{4}|x_k| > W$ and

$$\sup \left| \Big((1-\epsilon_1)\langle \pi', \hat{Z}^{x_k}(s)\rangle - \langle \pi, \hat{Z}^{x_k}(s)\rangle\Big) - \Big((1-\epsilon_1)\langle \pi', \hat{Z}(s)\rangle - \langle \pi, \hat{Z}(s)\rangle\Big) \right| \leq \frac{\epsilon_2}{4}$$

Thus for $k > K$, we have

$$(1-\epsilon_1)\langle \pi', \hat{Z}^{x_k}(s)\rangle - \langle \pi, \hat{Z}^{x_k}(s)\rangle \geq \frac{\epsilon_2}{4} \qquad \textit{for } s \in [t-\tau, t+\tau]$$

which is equivalent to,

$$(1-\epsilon_1)\langle \pi', Z^{x_k}(s)\rangle - \langle \pi, Z^{x_k}(s)\rangle \geq \frac{\epsilon_2}{4}|x_k| > W \qquad \textit{for } s \in [(t-\tau)|x_k|, (t+\tau)|x_k|] \tag{C.10}$$

(C.10) implies that

$$\langle \pi, Z^{x_k}(s)\rangle < (1-\epsilon_1)\langle \pi', Z^{x_k}(s)\rangle \qquad \textit{for } s \in [(t-\tau)|x_k|, (t+\tau)|x_k|] \tag{C.11}$$

and

$$\langle \pi', Z^{x_k}(s) \rangle > W \qquad \textit{for } s \in [(t-\tau)|x_k|, (t+\tau)|x_k|] \tag{C.12}$$

Let $t' \in [(t-\tau)|x_k|, (t+\tau)|x_k|]$ be the first time (since time $(t-\tau)|x_k|$) that the scheduling algorithm selects a matching $\pi' \in \Pi'(t)$ as it's output (we simply set $t' = (t+\tau)|x_k|$ if it never selects a matching $\pi' \in \Pi'(t)$ throughout this time interval). By (C.11), we have

$$t' - (t-\tau)|x_k| \leq \Delta\left(\lfloor (t-\tau)|x_k| \rfloor,\, \epsilon_1,\, \omega\right)$$

By (C.9), (C.11) and (C.12), the system should never select a matching $\pi \notin \Pi'(t)$ throughout time interval $[t', (t+\tau)|x_k|]$. Thus, for any $\pi \notin \Pi'(t)$ we have

$$\begin{aligned} T_\pi^{x_k}(u_2, \omega) - T_\pi^{x_k}(u_1, \omega) &\leq t' - (t-\tau)|x_k| \\ &\leq \Delta\left(\lfloor (t-\tau)|x_k| \rfloor,\, \epsilon_1,\, \omega\right) \end{aligned}$$

for $u_1 \leq u_2$, $u_1, u_2 \in [(t-\tau)|x_k|, (t+\tau)|x_k|]$. Therefore, for any $u_1, u_2 \in [(t-\tau), (t+\tau)]$ with $u_1 \leq u_2$, we have

$$T_\pi^{x_k}(u_2|x_k|, \omega) - T_\pi^{x_k}(u_1|x_k|, \omega) \leq \Delta\left(\lfloor (t-\tau)|x_k| \rfloor,\, \epsilon_1,\, \omega\right)$$

i.e.,

$$\hat{T}_\pi^{x_k}(u_2, \omega) - \hat{T}_\pi^{x_k}(u_1, \omega) \leq \frac{1}{|x_k|} \Delta\left(\lfloor (t-\tau)|x_k| \rfloor,\, \epsilon_1,\, \omega\right)$$

Taking the limit as $k \to \infty$ and by Lemma 10, we have

$$\hat{T}_\pi(u_2) - \hat{T}_\pi(u_1) = 0$$

for any $u_1, u_2 \in [(t-\delta),(t+\delta)]$ with $u_1 \leq u_2$, from which (C.7) follows, proving the lemma. □

Theorem 7 is implied by Proposition 4 and the following theorem.

Theorem 8. *Assume the switch works under a scheduling algorithm that satisfies both the* (ϵ, δ)*-MWM condition and the sublinear degradation condition. Then the fluid model is stable if we have*

$$\sum_{j=1}^{N} \lambda_{ij} < 1 \qquad i = 1, \ldots, N \tag{C.13}$$

$$\sum_{i=1}^{N} \lambda_{ij} < 1 \qquad j = 1, \ldots, N \tag{C.14}$$

Proof. The proof is similar to the proof of Theorem 1 in [45]. For completeness, we produce a full proof here.

Suppose $(\hat{Z}, \hat{D}, \hat{T})$ is a fluid limit path with $\hat{Z}(0) = 1$. Define a Lyapunov function $f(t) = \langle \hat{Z}(t), \hat{Z}(t) \rangle$. We have $f(t) \geq 0$ and $f(t) = 0 \Leftrightarrow \hat{Z}(t) = 0$. It's then sufficient to prove that $\exists\, t_0 \geq 0$ such that $f(t) = 0$ for $t \geq t_0$

For a permutation matrix π, define $w_\pi(t) = \langle \pi, \hat{Z}(t) \rangle$. Let $w(t) = \max_\pi w_\pi(t)$. Let λ be the $N \times N$ matrix with entries λ_{ij} (flattened into a N^2-dimensional vector in the row-major order). Under condition (C.13) and (C.14) λ is doubly substochastic and can therefore be written as a convex combination of permutation matrices [74, 75, 76], i.e., there exists constants $p_\pi \in [0, 1)$, $\pi \in \Pi$, such that

$$\lambda = \sum_{\pi \in \Pi} p_\pi \pi$$

$$\sum_{\pi \in \Pi} p_\pi < 1$$

Let $\delta = \frac{1}{2|\Pi|}(1 - \sum_{\pi\in\Pi} p_\pi)$. We have

$$\sum_{\pi\in\Pi} (p_\pi + \delta) < 1$$

By the definition of $w(t)$, for $t \geq 0$ we have

$$\begin{aligned}
w(t) &> \sum_{\pi\in\Pi} (p_\pi + \delta)\, w_\pi(t) \\
&= \sum_{\pi\in\Pi} (p_\pi + \delta)\, \langle \pi, \hat{Z}(t)\rangle \\
&= \langle \sum_{\pi\in\Pi} (p_\pi + \delta)\, \pi, \hat{Z}(t)\rangle \\
&= \langle \sum_{\pi\in\Pi} p_\pi \pi, \hat{Z}(t)\rangle + \langle \sum_{\pi\in\Pi} \delta\pi, \hat{Z}(t)\rangle \\
&= \langle \lambda, \hat{Z}(t)\rangle + \langle \sum_{\pi\in\Pi} \delta\pi, \hat{Z}(t)\rangle \\
&= \langle \lambda, \hat{Z}(t)\rangle + \delta\langle \sum_{\pi\in\Pi} \pi, \hat{Z}(t)\rangle \\
&\geq \langle \lambda, \hat{Z}(t)\rangle + \delta \sum_{i,j=1}^{N} \hat{Z}_{ij}(t) \\
&\geq \langle \lambda, \hat{Z}(t)\rangle + \delta \sqrt{\sum_{i,j=1}^{N} \hat{Z}_{ij}^2(t)} \\
&= \langle \lambda, \hat{Z}(t)\rangle + \delta \sqrt{f(t)}
\end{aligned} \tag{C.15}$$

Let t be a fixed value such that w and $\hat{Z}$ are differentiable at t. Let Π' be the set of matchings π such that $w_\pi(t) = w(t)$. Then we have $\frac{d}{dt} w_\pi(t) = \frac{d}{dt} w(t)$ for $\pi \in \Pi'$ (see, for example, the proof of Lemma 3.2 of [77]). Hence, by (C.4) and Lemma 11, we have

$$\sum_{\pi\in\Pi'} \frac{d}{dt} \hat{T}_\pi(t) = 1$$

By (C.3), it follows that

$$\begin{aligned}
\langle \hat{Z}(t), \frac{d}{dt}\hat{D}(t)\rangle &= \langle \hat{Z}(t), \sum_{\pi\in\Pi'} \pi \frac{d}{dt}\hat{T}_\pi(t)\rangle \\
&= \sum_{\pi\in\Pi'} \langle \hat{Z}(t), \pi \frac{d}{dt}\hat{T}_\pi(t)\rangle \\
&= \sum_{\pi\in\Pi'} w_\pi(t) \frac{d}{dt}\hat{T}_\pi(t) \\
&= w(t) \sum_{\pi\in\Pi'} \frac{d}{dt}\hat{T}_\pi(t) \\
&= w(t)
\end{aligned}$$

Thus, by (C.15),

$$\begin{aligned}
\frac{d}{dt}f(t) &= 2\langle \hat{Z}(t), \frac{d}{dt}\hat{Z}(t)\rangle \\
&= 2\left(\langle \hat{Z}(t), \lambda\rangle - \langle \hat{Z}(t), \frac{d}{dt}\hat{D}(t)\rangle\right) \\
&= 2\left(\langle \hat{Z}(t), \lambda\rangle - w(t)\right) \\
&< -2\delta\sqrt{f(t)}
\end{aligned}$$

It follows from Lemma 9 that $f(t) = 0$ for $t \geq \sqrt{f(0)}/\delta$, proving the theorem. □

REFERENCES

[1] C.-S. Chang, D.-S. Lee, and Y.-S. Jou, "Load balanced birkhoff–von neumann switches, part i: One-stage buffering," *Computer Communications*, vol. 25, no. 6, pp. 611–622, 2002.

[2] C.-S. Chang, D.-S. Lee, and C.-M. Lien, "Load balanced birkhoff–von neumann switches, part ii: Multi-stage buffering," *Computer Communications*, vol. 25, no. 6, pp. 623–634, 2002.

[3] W. Ding, J. Xu, J. G. Dai, Y. Song, and B. Lin, "Sprinklers: A randomized variable-size striping approach to reordering-free load-balanced switching," in *ACM CoNext, the 10th International Conference on Emerging Networking EXperiments and Technologies*, ACM, 2014.

[4] J. J. Jaramillo, F. Milan, and R Srikant, "Padded frames: A novel algorithm for stable scheduling in load-balanced switches," *Networking, IEEE/ACM Transactions on Networking*, vol. 16, no. 5, pp. 1212–1225, 2008.

[5] I. Keslassy, "The load-balanced router," PhD thesis, Stanford University, 2004.

[6] B. Lin and I. Keslassy, "The concurrent matching switch architecture," *Networking, IEEE/ACM Transactions on*, vol. 18, no. 4, pp. 1330–1343, 2010.

[7] I. Keslassy, C.-S. Chang, N. McKeown, and D.-S. Lee, "Optimal load-balancing," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, IEEE, vol. 3, 2005, pp. 1712–1722.

[8] I. Keslassy and N. McKeown, "Maintaining packet order in two-stage switches," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, IEEE, vol. 2, 2002, pp. 1032–1041.

[9] L. G. Valiant, "A scheme for fast parallel communication," *SIAM journal on computing*, vol. 11, no. 2, pp. 350–361, 1982.

[10] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, ACM, 1981, pp. 263–277.

[11] R. Aleliunas, "Randomized parallel communication (preliminary version)," in *Proceedings of the first ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, ACM, 1982, pp. 60–72.

[12] D. Mitra and R. A. Cieslak, "Randomized parallel communications on an extension of the omega network," *Journal of the ACM (JACM)*, vol. 34, no. 4, pp. 802–824, 1987.

[13] A. Singh, W. J. Dally, B. Towles, and A. K. Gupta, "Locality-preserving randomized oblivious routing on torus networks," in *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, ACM, 2002, pp. 9–13.

[14] A. Singh, "Load-balanced routing in interconnection networks," PhD thesis, Stanford University, 2005.

[15] M. Henrion, K. Schrodi, D Boettle, M De Somer, and M Dieudonne, "Switching network architecture for atm based broadband communications," in *Switching Symposium, 1990. XIII International*, IEEE, vol. 5, 1990, pp. 1–8.

[16] I. Keslassy, S.-T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown, "Scaling internet routers using optics," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, ACM, 2003, pp. 189–200.

[17] P. Bernasconi, J. Gripp, D. Neilson, J. Simsarian, D. Stiliadis, A. Varma, and M. Zirngibl, "Architecture of an integrated router interconnected spectrally (iris)," in *High Performance Switching and Routing, 2006 Workshop on*, IEEE, 2006, 8–pp.

[18] K. Argyraki, S. Baset, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, E. Kohler, M. Manesh, S. Nedevschi, and S. Ratnasamy, "Can software routers scale?" In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, ACM, 2008, pp. 21–26.

[19] M. Kodialam, T. Lakshman, and S. Sengupta, "Efficient and robust routing of highly variable traffic," in *In Proceedings of Third Workshop on Hot Topics in Networks (HotNets-III*, Citeseer, 2004.

[20] M. S. Kodialam, T. Lakshman, J. B. Orlin, and S. Sengupta, "A versatile scheme for routing highly variable traffic in service overlays and ip backbones.," in *INFOCOM*, 2006.

[21] R. Zhang-Shen and N. McKeown, "Designing a predictable internet backbone network," in *Proc. HOTNETS*, 2004, pp. 1–6.

[22] ——, "Designing a fault-tolerant network using valiant load-balancing," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, IEEE, 2008, pp. 2360–2368.

[23] H. Nagesh, V. Poosala, V. Kumar, P. Winzer, and M. Zirngibl, "Load-balanced architecture for dynamic traffic," in *Optical Fiber Communication Conference*, Optical Society of America, 2005, OME67.

[24] P. J. Winzer, F. B. Shepherd, P. Oswald, and M. Zimgibl, "Robust network design and selective randomized load balancing," in *Optical Communication, 2005. ECOC 2005. 31st European Conference on*, IET, vol. 1, 2005, pp. 23–24.

[25] F. Shepherd and P. Winzer, "Selective randomized load balancing and mesh networks with changing demands," *Journal of Optical Networking*, vol. 5, no. 5, pp. 320–339, 2006.

[26] W. Ni, C. Huang, J. Wu, and M. Savoie, "Availability of survivable valiant load balancing (vlb) networks over optical networks," *Optical Switching and Networking*, vol. 10, no. 3, pp. 274–289, 2013.

[27] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Towards a next generation data center architecture: Scalability and commoditization," in *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, ACM, 2008, pp. 57–62.

[28] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *ACM SIGCOMM computer communication review*, ACM, vol. 39, 2009, pp. 51–62.

[29] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *INFOCOM, 2013 Proceedings IEEE*, Turin, Italy, 2013, pp. 2130–2138.

[30] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 39, 2009, pp. 39–50.

[31] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks.," in *NSDI*, vol. 10, 2010, pp. 19–19.

[32] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 41, 2011, pp. 266–277.

[33] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, ACM, 2011, p. 8.

[34] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 51–62, 2007.

[35] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz, "Per-packet load-balanced, low-latency routing for clos-based data center networks," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, ACM, 2013, pp. 49–60.

[36] M. Shafiee and J. Ghaderi, "A simple congestion-aware algorithm for load balancing in datacenter networks," *IEEE/ACM Transactions on Networking*, 2017.

[37] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese, *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 44, 2014, pp. 503–514.

[38] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 465–478, 2015.

[39] J. F. Hayes and T. V. G. Babu, *Modeling and analysis of telecommunications networks*. John Wiley & Sons, 2004.

[40] K. Nakagawa, "On the series expansion for the stationary probabilities of an m/d/1 queue," *Journal of the Operations Research Society of Japan*, vol. 48, no. 2, pp. 111–122, 2005.

[41] R. Bhagwan and B. Lin, "Fast and scalable priority queue architecture for high-speed network switches," in *IEEE INFOCOM*, IEEE, 2000.

[42] A. Ioannou and M. G. Katevenis, "Pipelined heap (priority queue) management for advanced scheduling in high-speed networks," *IEEE/ACM Transactions on Networking*, vol. 15, no. 2, pp. 450–461, 2007.

[43] H. Wang and B. Lin, "Per-flow queue management with succinct priority indexing structures for high speed packet scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1380–1389, 2013.

[44] O. Rottenstreich, P. Li, I. Horev, I. Keslassy, and S. Kalyanaraman, "The switch reordering contagion: Preventing a few late packets from ruining the whole party," *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1262–1276, 2014.

[45] J. G. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE*

Computer and Communications Societies. Proceedings. IEEE, IEEE, vol. 2, 2000, pp. 556–564.

[46] J. G. Dai, "On positive harris recurrence of multiclass queueing networks: A unified approach via fluid limit models," *The Annals of Applied Probability*, pp. 49–77, 1995.

[47] J. Turner, "New directions in communications(or which way to the information age?)" *IEEE communications Magazine*, vol. 24, no. 10, pp. 8–15, 1986.

[48] H. Kung, T. Blackwell, and A. Chapman, "Credit-based flow control for atm networks: Credit update protocol, adaptive credit allocation and statistical multiplexing," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 24, 1994, pp. 101–114.

[49] C. Özveren, R. Simcoe, and G. Varghese, "Reliable and efficient hop-by-hop flow control," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 24, 1994, pp. 89–100.

[50] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 25, 1995, pp. 231–242.

[51] B. Bensaou, K. Chan, and D. H. Tsang, "Credit-based fair queueing (cbfq): A simple and feasible scheduling algorithm for packet networks," in *IEEE ATM Workshop 1997. Proceedings*, IEEE, 1997, pp. 589–594.

[52] S. T. Maguluri, R Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *INFOCOM, 2012 Proceedings IEEE*, IEEE, 2012, pp. 702–710.

[53] P. Giaccone, B. Prabhakar, and D. Shah, "Randomized scheduling algorithms for high-aggregate bandwidth switches," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 546–559, 2003.

[54] Q. Zhao, J. Xu, and Z. Liu, "Design of a novel statistics counter architecture with optimal space and time efficiency," *ACM SIGMETRICS Performance Evaluation Review*, vol. 34, no. 1, pp. 323–334, 2006.

[55] P. Robert, *Stochastic networks and queues*. Springer Science & Business Media, 2013, vol. 52.

[56] B. Li and R. Srikant, "Queue-proportional rate allocation with per-link information in multihop networks," in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMET-

RICS '15, Portland, Oregon, USA: ACM, 2015, pp. 97–108, ISBN: 978-1-4503-3486-0.

[57] J. Dai, "Stability of open multiclass queueing networks via fluid models," *IMA Volumes in Mathematics and Its Applications*, vol. 71, pp. 71–71, 1995.

[58] J. G. Dai, *Stability of fluid and stochastic processing networks*. University of Aarhus. Centre for Mathematical Physics and Stochastics (MaPhySto)[MPS], 1998.

[59] M. Bramson *et al.*, "Stability of queueing networks," *Probability Surveys*, vol. 5, pp. 169–345, 2008.

[60] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Transactions on Communications*, vol. 35, no. 12, pp. 1347–1356, 1987.

[61] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, 1992.

[62] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM*, vol. 19, no. 2, pp. 248–264, 1972.

[63] N. McKeown, "Scheduling algorithms for input-queued cell switches," PhD thesis, University of California at Berkeley, 1995.

[64] ——, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, 1999.

[65] M. Bayati, B. Prabhakar, D. Shah, and M. Sharma, "Iterative scheduling algorithms," in *Proceedings of the IEEE INFOCOM*, Anchorage, AK, USA, 2007, pp. 445–453.

[66] M. Bayati, D. Shah, and M. Sharma, "Max-product for maximum weight matching: Convergence, correctness, and lp duality," *IEEE Transactions on Information Theory*, vol. 54, no. 3, pp. 1241–1251, 2008.

[67] S. Atalla, D. Cuda, P. Giaccone, and M. Pretti, "Belief-propagation-assisted scheduling in input-queued switches," *IEEE Transactions on Computers*, vol. 62, no. 10, pp. 2101–2107, 2013.

[68] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Transactions on Communications*, vol. 47, no. 8, pp. 1260–1267, 1999.

[69] G. R. Gupta, S. Sanghavi, and N. B. Shroff, "Node weighted scheduling," in *Proceedings of the ACM SIGMETRICS*, Seattle, WA, USA, 2009, pp. 97–108.

[70] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," in *Proceedings of the IEEE INFOCOM*, San Francisco, CA, USA, 1998, pp. 533–539.

[71] D. Shah, P. Giaccone, and B. Prabhakar, "Efficient randomized algorithms for input-queued switch scheduling," *IEEE Micro*, vol. 22, no. 1, pp. 10–18, 2002.

[72] S. Ye, T. Shen, and S. Panwar, "An $O(1)$ scheduling algorithm for variable-size packet switching systems," in *Proceedings of the 48th Annual Allerton Conference*, Illinois, USA, 2010, pp. 1683–1690.

[73] L. Gong, P. Tune, L. Liu, S. Yang, and J. J. Xu, "Queue-proportional sampling: A better approach to crossbar scheduling for input-queued switches," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, p. 3, 2017.

[74] G. Birkhoff, "Tres observaciones sobre el algebra lineal," *Univ. Nac. Tucumán Rev. Ser. A*, vol. 5, pp. 147–151, 1946.

[75] J. v. Neumann, "A certain zero-sum two-person game equivalent to the optimal assignment problem," *Contributions to the Theory of Games*, vol. 2, pp. 5–12, 1953.

[76] I. Olkin and A. W. Marshall, *Inequalities: Theory of majorization and its applications*. Academic press, 2016.

[77] J. G. Dai and G. Weiss, "Stability and instability of fluid models for reentrant lines," *Mathematics of Operations Research*, vol. 21, no. 1, pp. 115–134, 1996.

VITA

Sen Yang received his B.S. degree in Electronic Engineering from Shanghai Jiao Tong University, China in 2010. He received his M.S. degree in Electronic and Communication Engineering of Shanghai Jiao Tong University and another M.S. degree in Electrical and Computer Engineering of Georgia Institute of Technology both in 2013. Sen joined School of Electrical and Computer Engineering, Georgia Institute of Technology, as a Ph.D. student in August 2013. His thesis work was conducted under the able guidance of Dr. Jun (Jim) Xu.